Reference Manual

7030 Data Processing System

IBM

Reference Manual

7030 Data Processing System

A22 6530 2

IBM 7030

IBM 7030 Printed in U.S.A. A22-6530-2

IBM®

International Business Machines Corporation

Data Processing Division, 112 East Post Road, White Plains, N.Y.

IBM Reference Manual

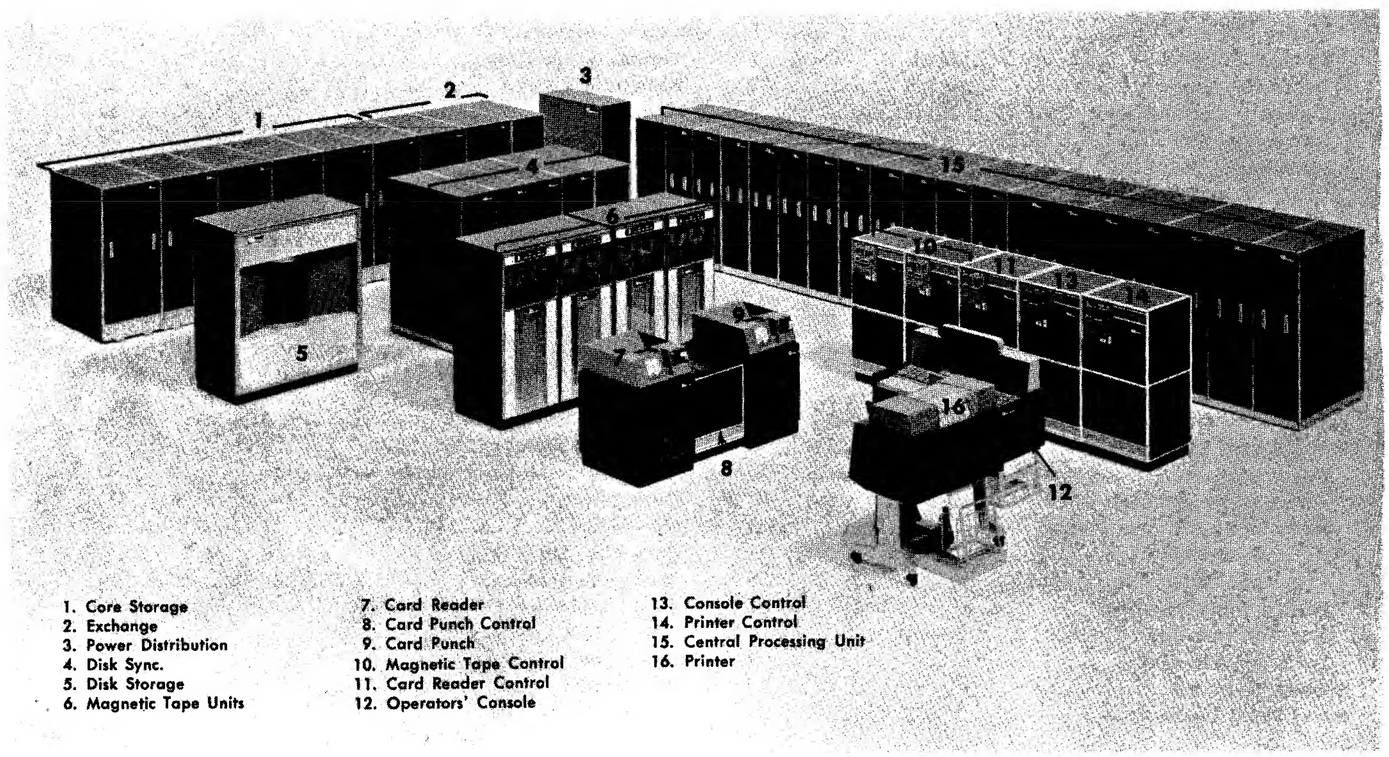
7030 Data Processing System

MAJOR REVISION August 1961

This edition, Form A22-6530-2, obsoletes Form A22-6530-1. Major changes occur on pages 33, 34, 66, 67, 83, 89, 91, 121, 125, 134, 135, 136, 141 through 148, and 157 through 162.

This description of the IBM 7030 Data Processing System is subject to modification by engineering developments.

Contents



IBM 7030 Data Processing System

IBM 7030 Data Processing System

The IBM 7030 Data Processing System substantially increases over-all performance on large technical computing problems. It solves important problems that earlier computers do not have the capacity and speed to complete in a reasonable time. The power and general-purpose design of the 7030 also permit the efficient solution of large data-processing and business-oriented problems, as well as a more rapid completion of many lesser problems.

Impressive speed increases have been realized in the design of new components and circuits in the 7030 system. To match and fully utilize this technological development, a high degree of overlapped operation, new input-output facilities, a more powerful instruction set, and other logical organization improvements have been made to reach the very high over-all performance of the system.

For example, the 7030 is equipped with the high-speed IBM 7302 Core Storage units. These units, when used with the 7030, will contain 16,384 words of 72 bits each. Several storage units can be cycled concurrently. Results produced can be placed in storage and one or more new instructions and operands can be fetched from storage, while the arithmetic unit is busy executing an instruction with data already in its registers. Indexing of instructions is carried on in a separate index arithmetic unit; some instructions can even be executed entirely in that unit while a previous operation is being completed in the main arithmetic section. Automatic interlocks and safeguards adjust the flow of information and assure that the program is executed correctly.

Because of the overlapped operation of many parts of the system, it is quite difficult to predict the exact speed at which a given program will be executed. The duration of each operation depends on the data, on the availability of various units, and on the instructions which preceded it. While a time can be given for each operation alone, in terms of a formula or as an average, the over-all time is not the sum of these times. Representative time estimates for a problem can only be obtained by simulating the many detailed time relationships in terms of that program or by actually running it. The computer is effective in adjusting the flow of information between various units during the execution of a program. Thus the programmer is relieved of the burden of optimizing his program.

Another factor contributing to high performance is the efficient handling of input-output and external devices. Extensive buffering and multiplexing facilities built into the 7030 exchange and associated channels permit the simultaneous operation of many input-output units together with computation. Eight input-output channels with an aggregate rate of about 6 million bits per second are provided with the exchange. These rates insure that high computational speeds can be maintained when dealing with data that cannot all be held in internal storage.

The input-output system is flexible enough so that almost any kind of device furnishing or accepting digital information can be connected to the computer.

Large high-speed magnetic disk storage units, each with multi-million word capacity, are available to store large data arrays and read or write at rates of a few microseconds per word.

The instruction set which has been developed for this system exhibits new and powerful features. As a result, fewer instructions are usually required to write a given program. Moreover, most of the instructions used in the inner loops of floating point arithmetic problems will be only half a word, or 32 bits, long. Fewer and shorter instructions mean less storage space for programs, fewer accesses to core storage, and fewer instructions executed. All of this contributes to high performance. Fewer instructions written also means less programming effort.

Much of the emphasis in the instruction set is on efficient floating point, indexing, and branching instructions, which form the heart of a great many programs in the technical field. It is equally important, though, to have available good instructions for housekeeping functions, for editing input-output data, and for processing data other than floating point numbers.

The use of solid-state components throughout makes the system inherently reliable. Automatic error detection, together with means for localizing faults and other maintenance aids, serves to reduce the time required to identify and repair those machine malfunctions which do arise. Automatic error correction is provided in areas where this is expected to improve performance materially. Automatic checking will provide continuous monitoring to insure that a reliable machine continues to be reliable. By reducing rerun time, checking enhances computer performance.

Another contribution to performance is made by features to improve operating techniques. Routine operating functions are, as much as possible, placed under direct control of the stored program. Input-output units require a minimum of manual set-up. Control panels are omitted entirely; all rearrangement and control of data is under the more general and flexible control of the stored program. Multiple input-output channels, buffering, and multiprogramming facilities make it possible to do independent input-output data conversion efficiently on units attached to the computer without significantly increasing the computing time on the major job. Physical switching of units or manual transfer of tape reels is thus avoided.

The console is separated from the main computer and becomes an optional input-output device. The keyboard, switches, lights, digital display, and console printer are all subject to programmed interpretation and control. The interpretive program may endow these devices with sophisticated control functions or may ignore them altogether. This interpretive approach to the console gives exceptional flexibility and makes possible console facilities which will not become inadequate as new operating techniques are developed. The use of programmed definition of console functions also permits protective and logging functions which would be quite uneconomical in hardware.

These facilities are provided to permit close communication between man and machine where human intervention and supervision is desired. These facilities are included because planned intervention and supervision can often bring a problem to completion more quickly than computation alone. Of course, matching a high performance computer directly to the thinking and reaction times of a human user is very costly in wasted machine power. With facilities provided for multiprogramming, however, the economics of human intervention may be radically changed. With two or more programs ready for operation, the machine need not wait idly while the user thinks. All console facilities are designed to take advantage of the intervention techniques made possible by multiprogramming.

System Organization

The basic system is composed of a central processing unit, one or more core storage units, an exchange, and input-output devices.

Information moves between the input-output de-

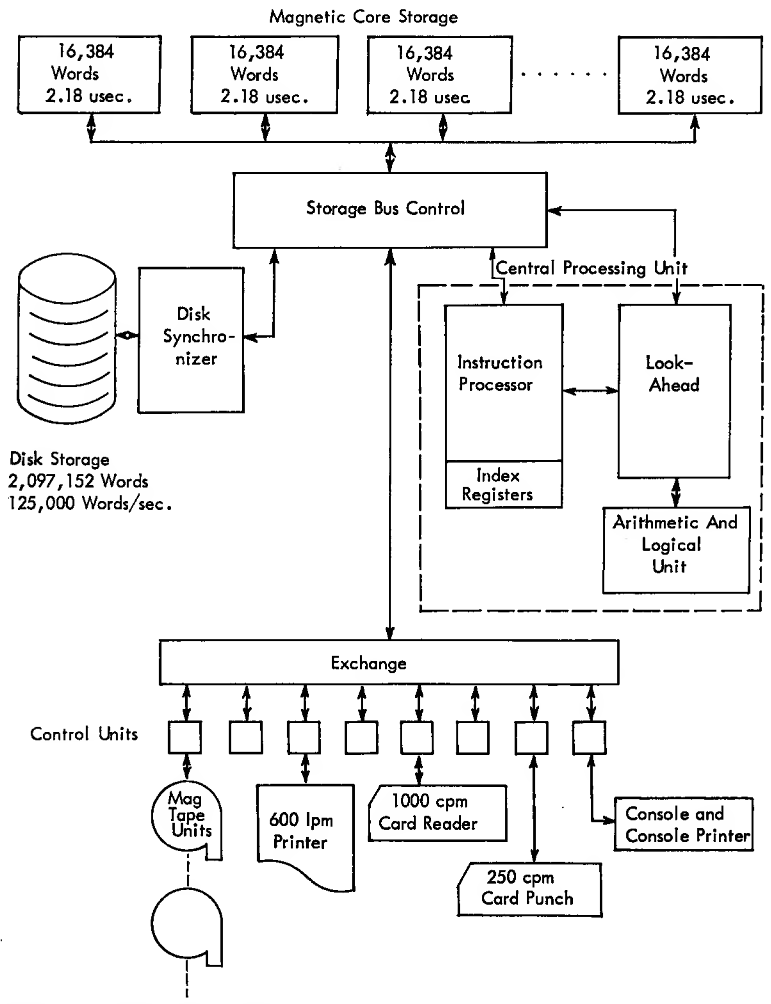


Figure 1. Typical IBM 7030 System

vices and the core storage under control of the exchange. The central processing unit consists of the registers, arithmetic devices, and control circuits necessary for performing operations upon data taken from core storage. The central processing unit is controlled by a succession of instructions which themselves come from core storage, and a special set of registers and controls permits the instructions to be obtained and modified at high speed.

A typical system configuration is shown in Figure 1.

Storage Units

Core Storage

The computing system uses IBM 7302 Core Storage units with a read-write cycle time of 2.18 microseconds. A word consists of 64 information bits and 8 non-addressable redundancy bits. Storage and bus action is checked, and any single-bit errors are automatically corrected without slowing the speed of operation.

The address space in instructions provides for addressing directly any of 2^{18} (262,144) word locations on any operations. Storage addresses are numbered from 0 to $2^{18}-1$ consecutively, although addresses 0 to 31 are reserved for index words and other special registers.

Each unit of storage consists of 16,384 (2^{14}) words. A typical system will contain several such units with each storage unit operating independently. In systems with two or more units, several storage references may be in process at the same time. In order that addressing may take advantage of this, successive addresses are distributed among different units. When a system includes two units, successive addresses alternate between the two. When a system includes four or more units, successive addresses are distributed within groups of four units.

Index Word Storage

The number of index registers used in the 7030 system is 16. A separate fast magnetic core storage with a read-out cycle of .4 μs is used for index words. The index words have addresses 16 to 31. These registers are located in the central processing unit for use by the instruction processor.

Special Registers

Many of the special registers of the machine are directly addressable to lend increased programming and operating flexibility to the system. Some of these registers are transistor registers; others are in a fast magnetic core storage. These special registers are assigned addresses 1 to 15.

Input and Output

Input to the system passes from the input devices to core storage through the exchange. The exchange assembles complete 64-bit words from the flow of input information and stores the assembled words in core storage locations without tying up the central processing unit. The CPU specifies the starting location and the number of input words to be read. While the CPU proceeds with computation, the exchange completes the input operation and signals the CPU when the transmission is finished.

The same exchange operates similarly for output, fetching core storage words and disassembling them for the output devices independently of the CPU. External storage devices, such as tapes, may be operated via the exchange as if they were input and output units.

The exchange can operate eight independent input-output units. This eight-channel exchange can be enlarged by adding other eight-channel groups. The exchange can address up to 32 channels. Each of the exchange channels has a nominal information rate of 500,000 bits per second. However, for special requirements, higher information rates are possible. The exchange as a whole can reach a peak rate of 100,000 words or 6,400,000 bits per second.

A wide variety of input-output units can be operated by the exchange. These include card readers and punches, printers, magnetic tape units, and operator's consoles. Most control units permit only one input-output device to be attached to a channel. The magnetic tape control unit, however, is designed to allow up to eight tape units to be attached to a single exchange channel. When this is done, of course, only one of these units can be operated at a time.

Many of the problems to be solved by the 7030 require auxiliary storage capacity and transfer rates in excess of those available in magnetic tape units to fully utilize the very high computational speeds of the 7030 system. To take care of this requirement, a large volume magnetic disk storage unit is available, capable of writing a full 64-bit word each 8 microseconds. Each disk unit has a capacity of 2,097,152 words. One or more disk units may be attached directly to the system by means of a disk synchronizer unit which functions like the exchange to control the flow of information to and from the disk unit by executing instructions specified by the central processing unit.

Using these disk units, large volumes of information can be continuously and rapidly transferred in and out of core storage without affecting the rates of any other input-output operations being carried on through the exchange or noticeably slowing computation in the CPU.

Central Processing Unit

The central processing unit consists basically of three functional units: the instruction processor, look-ahead, and the arithmetic and logical unit. The central processing unit performs arithmetic and logical operations upon operands taken from storage. Operations are specified one at a time by instructions, which are also taken from storage. Each instruction fundamentally specifies an operation and an operand. The operand specification is made up of an *address* and an *index word address.* Part of the index word contents are added to the address to obtain an *effective address.* The effective address actually designates the operand used. The additions needed to derive the effective address and to modify index words are performed in the instruction processor by an independent index arithmetic unit.

Instruction Processor

Since the arithmetic and logical unit operates very rapidly, it needs to receive data and decoded and modified instructions at high speed. The decoding and modifying are performed in the instruction processor, using the index arithmetic unit, while the arithmetic and logical unit is executing some preceding instruction. Several types of instructions can actually be executed in the instruction processor without requiring the use or time of the main arithmetic unit.

The index arithmetic unit consists of registers for holding instructions to be modified and the index words that are used in the modification. When index words themselves are modified, some of these registers also hold the operand data. The possible modifications to index registers include loading, storing, adding, and comparing. The index arithmetic unit includes gates for selecting the necessary fields in index and instruction words and a 24-bit algebraic adder. The index words themselves are addressable as locations 16 to 31 in a special high-speed core storage.

An instruction may be one word or one-half word in length. Full and half-length instructions can be intermixed without regard to word boundaries.

Instructions are taken in succession under control of an instruction counter. Alteration of the succession of instructions is possible by branching operations, which can be controlled by a wide variety of conditions. Automatic interruption of the normal sequence can also be caused by many conditions. The conditions for interruption and control of branching are represented by bits in an indicator register. The interruption system also includes a mask register for controlling interruption and a base address register for selecting suitable alternate programs. When needed, the address of the input or output unit causing an interruption can be read from a channel address register which can only be set up by the exchange.

The instruction counter contents can be stored by a specific operation. The indicator, mask, interruption address and unit address registers are assigned certain of the first sixteen addresses.

The interpretation and execution of instructions is also controlled by an address monitoring system. This provides two boundary registers which define a protected storage region. The boundary registers are also assigned certain of the first sixteen addresses.

Special control bits are provided for signalling other computers in multi-computer systems and for causing the machine to simulate malfunctions. These bits lie in certain of the first sixteen word addresses. Others of these special addresses are used for a time clock and an interval timer whose values are stored in a high-speed core storage.

Look-Ahead

The parallelism or overlap in the modification and execution of instructions is extended to include further overlapping of the fetching of instructions and data from core storage. In the previous description of the core storage units, mention was made of the fact that several core storage units can be referenced for instructions and data simultaneously. The device which coordinates and controls the overlapping of instruction processing is known as the Look-Ahead.

When the instruction processor has finished decoding and modifying an instruction, a fetch request is made to core storage for the data associated with this instruction. The instruction and its data are then loaded into one of four look-ahead levels, preparatory to being executed in turn by the arithmetic and logical unit. The arithmetic and logical unit operates very rapidly, much more rapidly than single instructions could be fetched from the relatively slow storage units. The instruction processor and look-ahead function together to maintain a reservoir of pre-processed instructions together with their data to allow the arithmetic and logical unit to operate at its maximum speed. From this point of view, look-ahead acts as a virtual storage for the arithmetic unit, effectively cutting the core storage reference rate to a small fraction of the actual time required for sequential storage accesses.

If the result of an arithmetic operation is to be returned to storage, it is first placed in look-ahead, from whence it returns to storage. Frequently, subsequent instructions will require data that are already in look-ahead, so that data references are eliminated in these cases.

In spite of the internal complexities, the overlapping of fetching, modification, and execution of instructions as well as the simultaneous fetching and storing of data does not place any constraints on the programmer. The only external difference in system operation due to look-ahead is a higher operation speed. The look-ahead unit takes care of the usual conditions which overlapping causes, and makes the machine appear as if it were dealing with only one instruction at a time.

Arithmetic Unit

The arithmetic unit consists of an apparatus for performing floating point arithmetic upon a data word in parallel and a closely associated mechanism for performing arithmetic and logical functions upon arbitrary fields of bits. In the latter case, the bits may be operated upon as individual entities or as numbers encoded in binary or decimal.

For simplicity, the arithmetic unit may be considered to be composed of four one-word registers and a short register. This conceptual structure is shown in Figure 2, where the full-length registers are labeled A, B, C, and D and the short register is labeled S. The registers marked A and B constitute the left and right halves of the accumulator. The registers marked C and D serve as storage registers, receiving words from core storage via look-ahead and assembling results to be placed in storage again by way of look-ahead. The short register stores the accumulator sign bit, data flag bits, and zone bits, collectively called the accumulator *sign byte.*

In floating point MULTIPLY and ADD, one factor comes from storage to register C while the other comes from the factor register (FT) to register D. The two are multiplied and added to the accumulator contents. In ordinary multiplication, one factor comes from storage to C and the other is taken from A to D. The product is developed in the cleared accumulator.

In variable field length operations, the core storage word or words containing the data are placed in C and D. The data are selected a few bits at a time and processed. The result either replaces the accumulator contents, or replaces selected bits of C and D whose contents are then returned to core storage. Binary multiplication and division data are stepped into the parallel mechanism a few bits at a time, then the actual operation is performed in parallel.

In division, the quotient appears in the accumulator and the remainder is developed in C and D from whence it returns to the remainder register (RR). Reg-

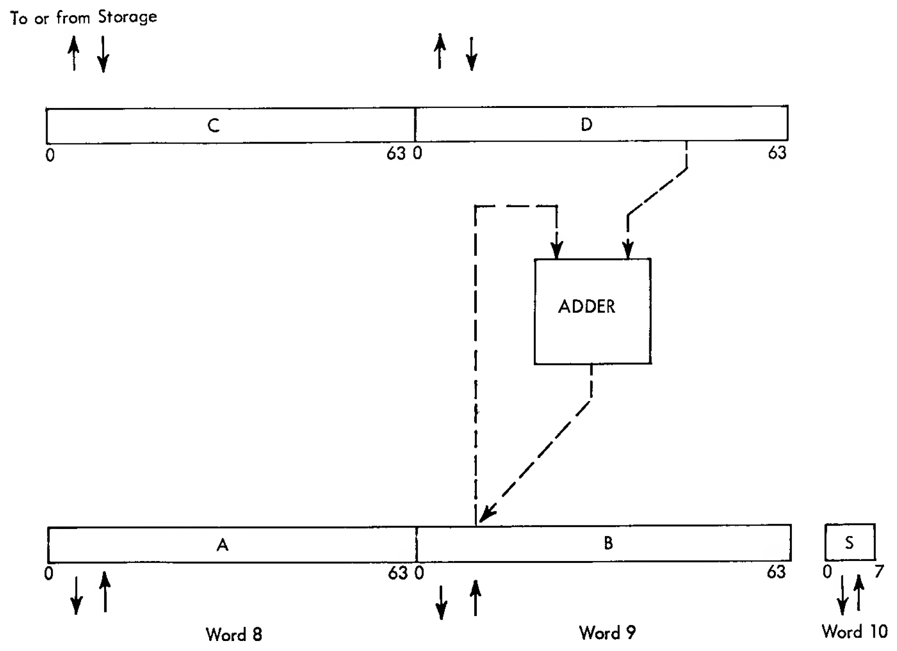


Figure 2. Conceptual Register Organization

isters A, B, FT, RR, and the sign byte register are assigned certain of the first sixteen addresses. Registers C and D always hold copies of the contents of storage words or of FT or RR and thus need no addresses. In some operations, other registers are used in the developing of results at high speeds, but the operation is always equivalent to that described above.

The transit register, a full-word core location, is used in some automatic subroutine entries. Two seven-bit registers are used in connective operations to hold bit counts developed upon the results. These, the all-ones counter and left-zeros counter, are also assigned certain of the first sixteen addresses.

Instruction Set

The instruction set is summarized in this section. Details will be found in the appropriate sections.

Classes of Operations

The operations available may be divided into these broad categories:

Floating point arithmetic operations
Integer arithmetic operations
Radix conversion operations
Connective operations
Index arithmetic operations
Branching operations
Transmission operations
Input-output operations
Miscellaneous operations

Summary of Data Arithmetic Operations

The arithmetic instruction set includes the elementary operations LOAD, ADD, STORE, MULTIPLY, and DIVIDE. Modifier bits are available to control the sign of the data. Thus the operations "subtract" or "add absolute" are obtained by use of sign modifiers with an ADD instruction, and are not provided as separate operations. These same modifiers permit controlling the sign of a number to be loaded, stored, multiplied, or divided.

A convenient feature of the MULTIPLY operation is that one of the factors is taken from the accumulator rather than a separate register and, therefore, may be the result of previous computation. Similarly, DIVIDE places the quotient in the accumulator, thus making it directly available for further operations.

Extensions of the basic set of arithmetic operations permit adding and counting in storage, rounding, cumulative multiplication, comparison, and other variations of the standard add operation.

One of these variations is a new add-type operation, ADD TO MAGNITUDE, that is like ADD when numbers of like signs are involved. With numbers of unlike signs, the ADD TO MAGNITUDE operation is different from a subtraction in that it does not allow the sign of the augend to change. Instead, the result is set to zero. This operation is useful when dealing with non-negative numbers or computing with discontinuous rates.

Most arithmetic operations are available in the floating point mode as well as the fixed point or integer mode. The floating point set includes additional instructions to handle portions of a floating point number and multiple precision numbers with ease. A floating point square root is provided.

Floating Point Arithmetic

Floating point arithmetic is done at very high speed in parallel binary arithmetic using a specialized data format wherein numbers are represented as a signed exponent, $\pm$ E (a power of 2), and a signed fraction, $\pm$ F, which together occupy a full 64-bit word.

The emphasis in the design of the floating point arithmetic is on high-speed computation of large mathematical problems. Simplification of the floating point instruction set is achieved through full utilization of the uniform nature of the floating point data. Two floating point instructions may be stored per word which results in an increase in storage efficiency and reduces the number of storage accesses.

Floating point instructions contain sign modifiers which permit any desired combination of operand signs. They also contain a normalization modifier which specifies the choice between normalized and unnormalized operation.

The 48-bit fraction makes it possible to compute in single precision mode for a number of problems which would have to be done in double precision on earlier computers. When required, however, the floating point instruction set contains a number of operations which provide double precision results. These operations have been designed to facilitate the programming of double or multiple precision operations.

In order to simplify significance studies, a mode of operation called "noisy mode" is provided in which results are altered in a specified fashion. Consecutive runs of the same problem in standard and noisy mode permit an estimate of the significance of the results to be obtained.

Floating point numbers cover a range between the positive and negative value of the fraction having the maximum exponent. Since the exponent range is finite, a discontinuity exists between the positive and negative values of the fraction having minimum exponent. Included in this range is the number zero. A control or flag bit has been incorporated in the exponent field to provide a straightforward control of data which exceeds the exponent range or falls within the range of discontinuity.

Integer Arithmetic

The class of integer arithmetic operations is designed to facilitate all data arithmetic on other than the specialized floating point numbers. The emphasis here is on versatility and economy of storage. Arithmetic may be performed directly in either decimal or binary radix. Individual numbers or *fields,* may be of any length, from 1 to 64 bits. Fields of different lengths may be assigned to adjacent locations in core storage, even if this means that a field lies partly in one word and partly in the next. Each field may be addressed directly by specifying its position and length in the instruction; the computer takes care of selecting the words required and altering only the desired information. Since the field length is explicitly stated in each instruction, rather than being implied by the data or by previous length-setting instructions, there is no restriction on the coding of variable field length data.

Individual characters, or *bytes,* in a field may also be varied in length. Thus a decimal digit may be compactly represented by a binary code of 4 bits, or it may be expanded to 6 or more bits when intermixed with alphabetic information. Decimal arithmetic may be performed directly on a decimal number, regardless of how many bits are used to encode a digit. Because decimal digits, alphabetic characters, and other single symbols may be encoded several different ways, the term *byte* is introduced to denote a single group of bits processed together, regardless of the meaning. A field may consist of one or more bytes.

The name integer arithmetic derives from the fact that in multiplication and division the results are normally aligned as if the numbers were integers. It is possible, though, to specify that numbers be offset so as to obtain any desired alignment of the radix point. An offset can be specified in every instruction, and there is no need for separate instructions to shift the contents of the accumulator.

Numeric data may be signed or unsigned. For unsigned data, the sign is simply omitted in storage, thus saving space and avoiding the task of assigning signs where there are none to begin with. Unsigned numbers are treated arithmetically as if they were positive.

A significant feature of the integer DIVIDE operation is that it will produce meaningful results provided the magnitudes of the dividend and the divisor fall within the bounds of numbers generally acceptable to the arithmetic unit. The only and obvious exception is a zero divisor. This greater freedom eliminates much of the scaling previously required before a DIVIDE instruction could be accepted.

Alphabetic and other non-numeric fields of various lengths may be handled by integer arithmetic operations as if they were unsigned binary numbers, regardless of the character code or the number of bits used for each character. In fact, there is no fixed character code built into the computer. Alphameric high-low comparisons are made by a simple binary subtraction of two fields. The only requirement is that the binary numbers representing each character fall into the comparing sequence desired for the application. If the code used for input or output does not conform to this comparing requirement, special provisions facilitate translating the code to any other form by programming a table look-up.

Another use for the integer arithmetic operation is to perform general arithmetic on portions of floating point words, instruction words, or index words. The floating point and index arithmetic instruction classes do contain addition and comparison instructions to cover the most frequent cases; the integer operations provide a complete set for all purposes.

An operation is provided that causes an automatic entry to a subroutine. A field of this instruction may be used to distinguish up to 128 pseudo-operations.

All integer operations are available in either decimal or binary form by setting one modifier bit. Decimal multiplication and division, however, are not built into the computer directly; their operation codes are instead used to cause an automatic entry to a subroutine which can take advantage of high-speed radix conversion and binary multiplication or division. Decimal multiplication and division are thus as convenient to program as if they had been built in, and they are faster this way.

The integer instruction set is similar to the floating point set; most of the operations in both sets have the same names and analogous meanings.

Radix Conversion

A group of radix conversion operations is provided to facilitate the use of decimal input and output while retaining the advantages of binary operation within the machine. These operations are also used in implementing the decimal multiplication and division pseudo-operations mentioned in the preceding section.

Several operations are provided to allow a variety of locations for the operand and the result. A field from storage may be converted and placed in either the accumulator or the transit register. Alternatively, a field from the accumulator may be converted and the result returned to the accumulator. In all these operations the operand is an integer, and it may be converted either from binary to decimal or from decimal to binary.

Connectives

Instructions which logically combine bits by AND, OR, and EXCLUSIVE OR have been included in earlier computers. These and many other non-arithmetic data handling operations are here replaced in a simple and orderly fashion by *connective* operations which provide many logical facilities not previously available. The operations are called CONNECT, CONNECT TO MEMORY, and CONNECT FOR TEST.

Each connective operation specifies a storage field of any length from 1 to 64 bits, as in integer arithmetic. Each bit in the storage field is logically combined with a corresponding bit in the accumulator. The result replaces the accumulator bit in CONNECT and the storage bit in CONNECT TO MEMORY. These operations make available certain tests and counts of zero and one bits. A new operation, CONNECT FOR TEST, has been added, which is analogous to COMPARE. This instruction allows fields to be tested and the corresponding counts to be made available without altering the content of either operand.

There are sixteen possible ways in which to combine or connect two bits. Each of these connectives can be specified with each of the three connective operations. Besides the connectives AND, OR, and EXCLUSIVE OR, there are connectives to match bits, to replace bits, and to set bits to zero or one. Either or both of the operands may be inverted.

While the term *logical connectives* suggests evaluation of elaborate expressions in Boolean algebra, the connective instructions have important everyday applications, such as assembling and converting input-output data. Their power lies in the ability to specify fields of any length and in any position in storage, whether they be single test bits or strings of adjacent bits.

The connective operations also facilitate new programming techniques. For example, instead of searching through a list of items to find the first one which is not zero, it is often faster and more convenient to maintain a list of yes-no bits. A set of 64 such bits may be tested with a single connective instruction which provides a count to indicate the position of the first non-zero bit on the left. This count is easily converted to an address for indexing to the location of the desired item. A second count gives the total number of one bits in the result field.

Index Arithmetic

Every instruction may have its address part modified by adding a number in a specified index register before using the address. Normally both the instruction and the index register remain unchanged. To alter the index registers is the function of the index arithmetic operations.

The set includes operations for loading, storing, incrementing, and comparing index values. The index value is a signed number and additions are algebraic. One of the instructions allows up to 16 index values to be added together for use in further indexing. Another indexing instruction provides the function of indirect addressing.

Each index word contains a count to keep track of the number of times a program loop has been traversed. Counting may be coupled with incrementing the index value. A third field in each index word specifies a *refill* address from which another index word may be loaded automatically.

Together these three fields provide a very convenient indexing technique. At each traversal of a program loop, a signed increment is added to the index value and the count is stepped down by one. When the count reaches zero, the index register is reset by refilling it from the storage location which contained the original value and count. All this may be done with one indexing instruction at the appropriate point in the loop.

The instruction set permits many other indexing techniques. An important one is the use of the refill address to indicate the next index word in succession in an indexing *chain*. Such chains permit the computation to progress through a series of items or records which are not stored in the order in which they are to be used. Chaining can greatly simplify insertion, deletion, and sorting of items by not requiring rearrangement of the data in storage.

Instructions generally specify one of a set of 15 index registers for address modification, but the number of available registers may be readily supplemented by other index locations in storage through an operation called RENAME. This operation identifies one designated index register with one of these storage locations and does the bookkeeping necessary to cause this storage location to reflect changes in the index register.

While indexing instructions are provided to change index values and counts explicitly, it is possible to use another mode, called *progressive indexing,* in which the index quantities may be advanced each time they are used. This mode may be applied to advantage for stepping along a string of data of various lengths without requiring a separate incrementing instruction at each step.

Branching

The branching operations either conditionally or unconditionally alter the instruction counter so as to change the course of a program. The number of operations is not large, but modifiers are available to provide a great deal of flexibility.

All machine state indicators such as sign, overflow, error, and input-output conditions are collected in one 64-bit indicator register. The BRANCH ON INDICATOR instruction may specify any one of these 64 indicators as the condition to be tested. A modifier specifies whether branching is to occur when the indicator is on or off. Another modifier may cause the tested indicator to be reset to zero.

A second operation, BRANCH ON BIT, permits testing a single bit anywhere in storage with one instruction. The tested bit may also be modified. This instruction places a virtually unlimited number of indicators under the direct control of the program.

Among the unconditional branches is a BRANCH RELATIVE operation which causes the address part to be added to the current contents of the instruction counter, so as to simplify relative addressing.

A hybrid operation combines advancing of an index word with testing and branching. Thus, the most common program loops may be closed with one half-length instruction.

Branch instructions may be coupled with an operation to store the instruction counter contents at any desired location before branching. This simplifies re-entry to a program from a subprogram.

Internal Data Transmission

The operation TRANSMIT provides the facilities to move a block of data from one set of addresses to another. One use of this operation is to preserve the contents of addressable registers, including index registers, in storage when it is necessary to bring in another program, and later to reload those registers to restore them to their earlier state before restarting the interrupted program.

A second operation, SWAP, interchanges the contents of two storage areas.

Input-Output

There are basically two operations for controlling input-output and external storage units: READ and WRITE. Each instruction specifies the unit desired and a storage area for the data to be read or written.

The storage area is specified by giving the address of a control word which contains the first data address in storage and a count of the number of words to be transferred. The control word also contains a refill address which can specify the address of another control word. Control words can thus be chained together to define storage areas which are not adjacent.

Control words have the same format as index words and can be used for indexing. This important feature means that the same word can first be used for reading new data, then for indexing while processing those data, and finally for writing the data from the same storage area. This technique greatly speeds up the processing and sorting of large files.

Various modifications of READ and WRITE are provided to fit different circumstances. Other instructions perform external controlling and switching functions which do not cause data to be transferred.

All instructions for operating external units are issued by the computer program but are executed independently of the program. A number of data transfers can thus take place simultaneously, all sharing access to storage. Signalling functions inform the program when each external process is completed.

All external units, regardless of their characteristics, are controlled by the same set of instructions. They are distinguished by a number assigned to each unit. Still, the complete set of functions needed to control external devices is rather complex. It is rarely necessary, though, to take care of all conditions relating to external units in every program. In a specific set of applications, it is possible to have a common supervisory program to take care of most of the exceptional conditions in a predefined manner, thus greatly reducing the amount of information to be specified in each operating program.

The apparent complexity of input-output functions is really an unavoidable consequence of the complex environment in which a powerful computer must operate. By using a supervisory program, rather than restrictive built-in controls, the flexibility is retained to adapt the computer to a wide range of applications.

Miscellaneous Operations

The miscellaneous category includes: EXECUTE, NO OPERATION, and BRANCH ENABLED AND WAIT.

EXECUTE gives the address of an instruction which the computer then executes. It permits tracing of programs at high speed.

By changing a single bit in the operation code, a branch instruction can be converted to NO OPERATION, and vice versa. This technique is a convenient way of altering the future course of a program.

BRANCH ENABLED AND WAIT replaces the familiar STOP instruction. Although the computer ceases to process instructions after setting a new value into the instruction counter, the interruption system is enabled and it is ready for an external signal to restart the program at any time. The start signal may come from an operator or from an external mechanism.

Since the computer may be set up to execute several programs on a time-shared basis, it is important that the computer be able to continue after one of the programs has come to an end or has run into trouble. BRANCH ENABLED AND WAIT is a program stop; there is no machine stop in the repertoire.

Features

New programming features not identified with specific instructions are summarized below.

Addressing

In instructions where this is meaningful, the position of a single bit in any word of storage can be addressed directly. A complete word and bit address forms a 24-bit number. The word address (18 bits) is on the left and the bit address (6 bits) is on the right of that number. For the purpose of bit addressing, the entire core storage can be considered as a set of consecutively numbered bits. Since the number of bits in a word (64) is a power of 2 and all addressing is binary, the address of the last bit of one word is followed immediately by the address of the first bit of the next word. If appropriate, word boundaries may be ignored by the program.

Other instructions use only full words as data, and these provide space for only 18 bits of address. The bit address is assumed to be zero. Still other instructions refer to half-words and use 19 bits of address. The extra bit is immediately to the right of the word address and the remaining five bits of the bit address are treated as zeros.

Index words provide space for sign and 24 bits in the value field, so that all addresses may be fully indexed to the bit level. The entire signed 24-bit address, with zeros inserted where instructions have fewer address bits, participates in the algebraic addition during modification. Where less than 24 bits are needed in the effective address, the low-order bits are dropped.

The ability to address and index to any bit position in storage is a powerful new programming feature. Data need not be unpacked and packed to fit an arbitrary word length imposed by storage for structural reasons. Data can be addressed directly at the location at which they are stored for input-output handling. Storage space can be assigned for greatest efficiency. For example, tables of character codes may be stored most conveniently in consecutive 8-bit locations, rather than taking a full word for each entry. Look-up of such a table involves merely offsetting the argument to attach three zero bits on the right and indexing this number with the starting address of the table.

Most of the internal machine registers are directly addressable. For instance, the accumulator may be added to itself by giving its address as the operand of an ADD instruction. An important use of the register addressability is for preserving and restoring the contents of internal registers by transmitting them as a block to or from some storage area with one TRANSMIT instruction.

Instead of selecting a location from which to fetch data, a modifier bit in the instruction may be set to specify that the address itself serve as data in the operation. It is then called an *immediate address*. Such data are limited to 24 bits. This feature is very convenient for defining short constants without having to provide the space and time for separate access to storage.

The term *direct address* is used to distinguish the usual type of address which defines the location of data for an operation or of an instruction to be executed.

The term *indirect address* refers to an address which defines the location of another address. An indirect address may select an immediate address, a direct address, or another indirect address. Indirect addresses are obtained in this system by the instruction LOAD VALUE EFFECTIVE which places the effective address found at the specified storage location into an index register for indexing on a subsequent instruction. Multiple-level indirect addressing is obtained when LOAD VALUE EFFECTIVE finds at the selected location another instruction LOAD VALUE EFFECTIVE which causes the indirect addressing process to be repeated.

Program Interruption

A program interruption system is provided for two quite distinct purposes. The first of these is to provide a means by which a computer can respond rapidly to extra-program circumstances which occur at arbitrary times, performing useful work while waiting for such circumstances. These circumstances will most often be signals from the exchange that some interrogation has been received or that an input-output operation is complete. For efficiency in real-time operation, the computer must quickly respond to these signals. This demands a system by which such signals cause a transfer of control to a suitable special program.

The second purpose is to permit the computer to make rapid and facile selection of alternate instructions when program-activated indicators signal that special circumstances have occurred. For example, it is clearly desirable to have such a system for arithmetic overflow since the alternatives are tedious and wasteful programmed testing or a costly machine stop when the condition arises. As another example, it is desirable to have a special routine seize control and take corrective steps whenever the regular program attempts a division by zero.

These two purposes, response to asynchronously occurring external signals and monitoring of exceptional conditions generated by the program itself, are distinct, and it would be conceivable to have systems for handling each independently. However, a single system serves both purposes equally well, and provision of a single uniform system permits more powerful operating techniques.

The program interruption system consists of a common indicator register which is continuously monitored. When one of the indicators comes on, the computer selects an instruction from a corresponding position in a table of correction instructions. This instruction is sandwiched into the program currently being executed at whatever time the interruption occurs. The extra instruction is commonly a STORE INSTRUCTION COUNTER IF BRANCH operation which leads to a correction routine while preserving the point at which the current program was interrupted. The table of correction instructions may be placed anywhere in storage.

Means are provided to select which indicators may cause interruption and when interruption will be permitted. Priorities can thus be established. If more than one interrupting condition should occur at one time, the system takes them in order. Special provisions are made to permit interruptions to any level to occur without causing program confusion.

The program interruption system was designed so that programming the base program will be straightforward, efficient, and as simple as the inherent conceptual complexities allow. The computer is not retarded by the interruption system, except when interruptions do in fact occur. Once this happens, and the computer has entered a special program pertaining to the interrupting condition, the full flexibility of a stored program computer may be brought into play. Elaborate correction programs will usually be written by specialists and are no burden on the programmer of the base program.

Address Monitoring

Address monitoring facilities are provided for two reasons. One is to make it possible for a program supervising the check-out of another program to detect when reference is made to a storage location outside the area assigned to that program. Another is to protect one program from accidental destruction by another program being executed on a multiprogrammed basis while the first program is waiting for service.

The upper and lower boundaries of the storage area to be defined are placed in two address boundary registers. An alarm will be given when an address falls either inside or outside the defined area, whichever is desired. Storing in protected areas is normally suppressed.

The addresses to be compared against the boundaries are the effective addresses after indexing, if any. Because it is often very difficult to predict all the addresses which might be generated by indexing or other address modification, especially when the program is not yet known to be free of errors, the built-in address monitoring facilities give far better protection than is possible by screening the program before execution.

Clocks

An interval timer is built in to measure elapsed time over relatively short intervals. It can be set to any value at any time, and an indicator shows when the time period has ended. This indicator can be used to cause an automatic program interruption.

To provide a continuous indication of time, a time clock is also furnished. This clock runs continuously while the machine is in operation, and its setting cannot be altered by the programmer. It may be used to time longer intervals for logging purposes, or, in connection with an external calibrating signal, to provide a time-of-day indication.

Operand Designation

Information Format

The computer system stores information in core storage in 64-bit words. Information transmission between storage, exchange and computer sections, is in parallel, a 64-bit word at a time. Information words of 64 bits are part of 72-bit machine words. The extra 8 bits are check bits, not available to the programmer, which permit single-error correction and double-error detection. Within the central processing unit, the information bits are combined with check bits in a variety of ways, in order to insure single error detection.

Words in storage are specified by a contiguous set of addresses. Addresses are numbered from 0 to 262,143 ($2^{18}-1$). In each word, the bits are numbered from 0 to 63, left to right. Bit 63 of a word may be considered adjacent to bit 0 of the next higher addressed word.

The set of addresses includes the addresses of registers located in the computer. By addressing such a register, its information may be treated as if it were located in core storage. A list of register addresses is shown in Figure 3, and these registers are described in the storage assignment section.

Storage location 0 is not available as standard storage. When information is taken from location 0, an all-zero word is obtained at all times. Information stored at location zero cannot be recovered.

Location		Name	Length	Bit Address
0	ext	Zero	64	0-63
1	ind	Interval Timer (P, a)	19	0-18
1	ind	Time Clock (P, b)	36	28-63
2	ext	Interruption Address (P)	18	0-17
3	int	Upper Boundary (P)	18	0-17
3	int	Lower Boundary (P)	18	32-49
3	int	Boundary Control Bit (P)	1	57
4	ext	Maintenance Bits	64	0-63
5	int	Channel Address (b)	7	12-18
6	int	Other CPU	19	0-18
7	int	Left Zeros Count	7	17-23
7	int	All Ones Count	7	44-50
8	int	Left Half of Accumulator	64	0-63
9	int	Right Half of Accumulator	64	0-63
10	int	Accumulator Sign	8	0-7
11	int	Indicators (c)	64	0-63
12	int	Mask (d)	64	0-63
13	ext	Remainder	64	0-63
14	ext	Factor	64	0-63
15	ext	Transit	64	0-63
16-31	ind	Index Registers X0-X15	64	0-63

P	=	Permanently protected area of storage
a	=	Read-only except for Store Value, Store Count, Store Refill, and Store Address
b	=	Read-only
c	=	Bit positions 0-19 are read-only
d	=	Bit positions 0-19 are always ones and positions 48-63 are always zeros
ext	=	External storage location
ind	=	Index core storage location
int	=	Internal registers

Figure 3. Register Addresses

Storage locations 1-15 contain various special registers of the central processing unit. These registers contain different numbers of bit positions as indicated in Figure 3. All other bits in locations 1-15 are always zero. They act like the bits of location 0.

The purpose and function of the index registers in storage locations 16-31 are explained under "Indexing."

Addresses greater than 31 are used for main core storage.

In some operations, information is processed a word at a time. In another group of operations, information is processed a half word at a time. A half word contains 32 bits and corresponds either to the first half, bits 0-31, or the second half, bits 32-63, of a full word. In a third group of operations, the variable-field-length operations, information is processed a field at a time. A field may have any length from one to 64 bits. A field may start at any bit position in a word in storage and continue through that word and into the word in the next higher addressed storage location. Thus, fields of any length from one to 64 bits may be stored in adjacent storage positions, regardless of core storage word boundaries.

Information, whether a full-word, half-word or field, is always addressed by the leftmost bit. The length, in bits, of the information is either implied by the operation to be performed or specifically stated as part of the instruction.

Data Format

Data generally represent alphameric information. For both arithmetic operations and non-arithmetic operations, the format of the data needs further specification besides the length of the field.

Arithmetic instructions specify either binary or decimal integer arithmetic, or binary floating point arithmetic. The adder used in the arithmetic operation adjusts its mode of operation accordingly.

In integer arithmetic, numbers can be signed or unsigned as specified by the instruction. When present, the sign of a number is specified by a single bit. A zero indicates a positive sign; a one indicates a negative sign.

With the sign, up to three flag bits may be used.

Flag bits permit special identification of numbers. Also, from zero to four zone bits may be used with the sign. The zone bits have no special function, but may be used as code bits to obtain an alphameric representation of the sign. Flag bits are placed to the right of the sign bit, zone bits to the left of the sign bit. Zone bits, sign bit, and flag bits are treated as a single information unit, or *byte*. The sign byte can be from one to eight bits long.

For unsigned numbers, the sign is assumed to be plus. During arithmetic operations, all numbers are handled as if they were signed. When unsigned operands are obtained from storage, the implied sign is supplied as specified by the instruction. Thus, an operand can be brought in without sign and the result stored with sign, or vice versa.

In binary integer arithmetic, the data fields specified have variable field length. For unsigned data, the entire field is treated as a positive binary integer. For signed data, the low-order byte is interpreted as a sign byte. The number of bits in the sign byte, the *byte size,* must be specified in the arithmetic instructions.

In decimal integer arithmetic, the digits 0 through 9 are represented by the ten four-bit binary integers 0000 through 1001. To these four bits, extra high-order bits (zone bits) may be added to permit alphameric representation. A maximum of four zone bits can be used, which makes the decimal character size, or byte size, a maximum of eight bits. Decimal arithmetic makes use of variable length fields. For unsigned data, the entire field is treated as a group of numeric bytes. All bytes have the same byte size, with a possible exception of the high-order byte. For signed data, the low-order byte is interpreted as a sign byte. The sign byte has the same size as the numeric bytes. Figure 4 shows possible data formats.

Decimal information placed in the accumulator always has byte size 4. The zone bits of decimal bytes do not enter the accumulator or the accumulator sign byte register. The zone bits of the sign byte register may be used to set the zone bits of decimal bytes in storage.

A standard data format is provided for floating point arithmetic (Figure 5). The field corresponds to a core storage word. It has a length of 64 and starts at bit address 0. The two portions of the floating point word are: left, the 12-bit exponent; and right, the 52-bit fraction. Both of these are signed binary numbers. The exponent has a flag bit, ten numeric bits, and a one-bit sign byte. The fraction has 48 numeric bits and a four-bit sign byte. The three flag bits of the fraction serve for the entire floating point word.

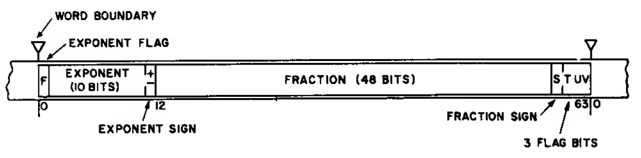


Figure 5. Floating Point Data Format

Floating point instructions may specify absolute value arithmetic. In absolute value floating point operations, the sign bit of the fraction is ignored. The remainder of the fraction sign byte and the exponent sign are used as in the signed arithmetic operations.

Byte size, field length, and bit address need not be specified in floating point operation since a standard format is used.

The programmer has the option to specify normalized or unnormalized operation. These two modes differ in the way in which the exponent and fraction of the result of an operation are adjusted.

For the non-arithmetic operations of the connect type, it is possible to specify the byte size of the data. The accumulator byte size is always eight for connect operations. The accumulator sign byte register is not used as part of the accumulator in these operations. The byte size of the data in storage may be one through eight.

Programming Note

The choice of byte size in decimal arithmetic and connect operations permits convenient byte expansion and contraction in code translation.

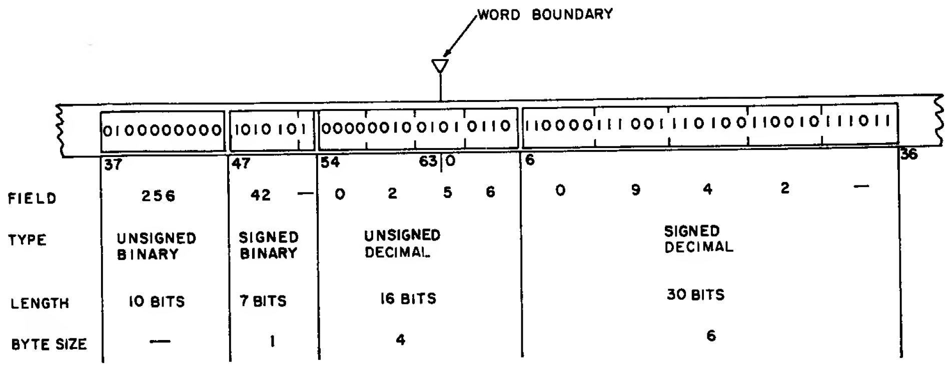


Figure 4. Data Formats

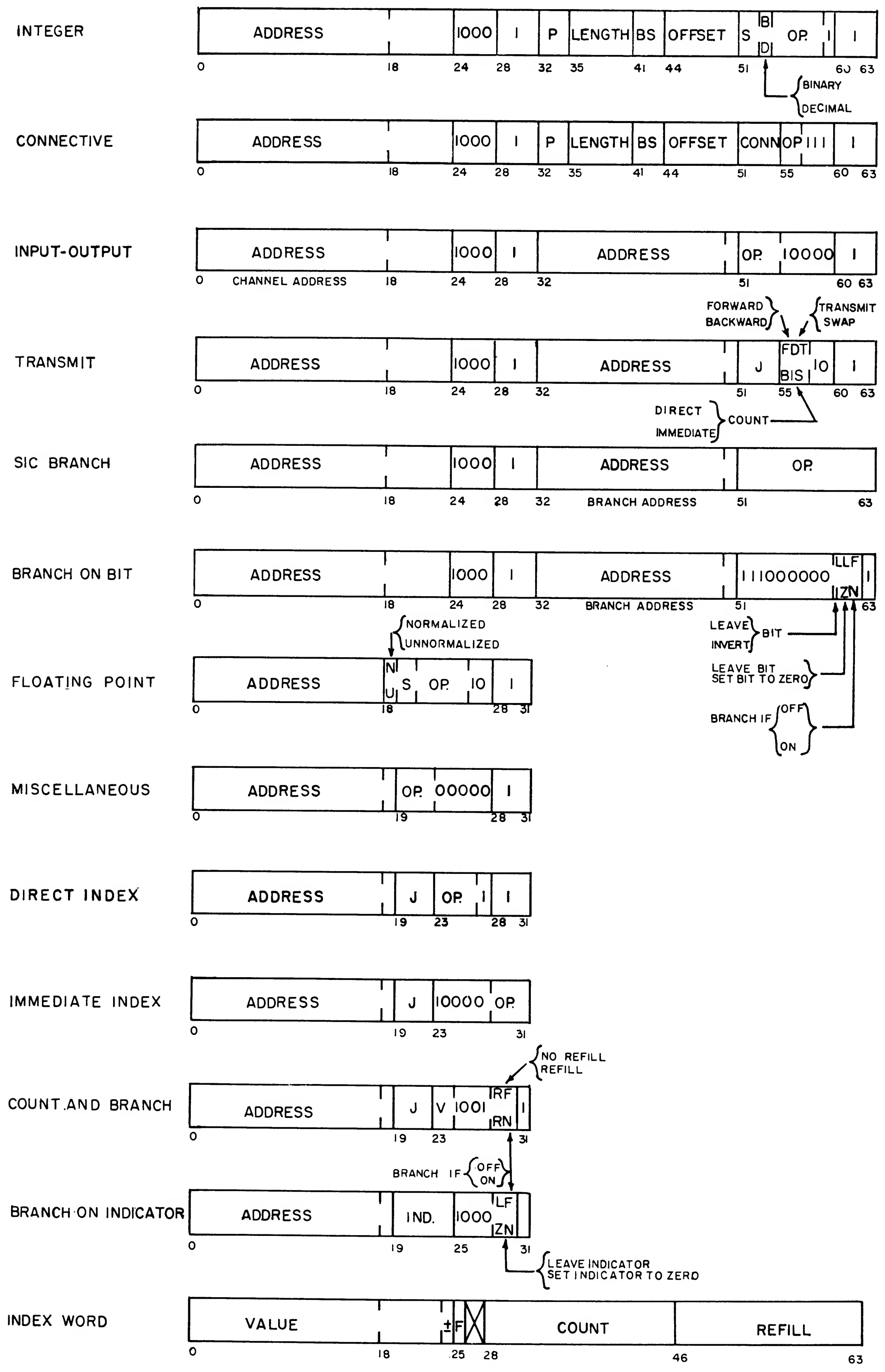


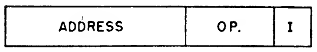
Figure 6. Instruction Formats

Control Format

Computer control is provided by instructions and indices. These are used in the control section of the computer to specify the operations to be performed. They are stored in core storage. When desired, they may be operated upon as data. Instructions occupy one or two half-words. They are referred to as half-length and full-length instructions. A full-length as well as a half-length instruction may have a bit address of 0 or 32. An index word always occupies a full word. It always has bit address 0. The formats used for instructions and indices are shown in Figure 6. In this figure, the bit address of all instructions is assumed to be 0. It equally well might have been 32. If the bit address is 32, the addresses of the individual fields in the instruction will be different from those shown. For convenience of presentation, all references to bits and fields will use the numbering of Figure 6.

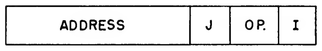
Instruction Format

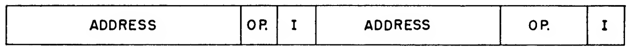
Instructions follow a basic pattern upon which some variations are made. This pattern consists of three parts: the address part, the operation part, and the index part.



The address part is leftmost in the instruction. It contains 18, 19 or 24 bits according to the type of operation to be performed. The address specifies the location of the operand which is involved in the operation specified by the instruction. The operation to be performed is specified in the operation part of the instruction. This part contains from 4 to 12 bits. The bits are used as class bits, to specify format and operation class; as code bits, to specify basic operations; and as modifier bits, to specify modifications to the basic operations. The operand address can be modified by the index specified in the rightmost part of the instruction half-word. The index part, if present, contains one or four bits. It may specify the address of one out of 15 words available for index operations.

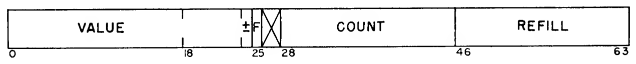
The instruction format as described so far occupies a half-word and specifies a single address operation with simple indexing. There are two major variations to this basic pattern. One includes a second index address. The other is the full-length format in which the half-word pattern is repeated. The operand address of the second half-word may be used as a second address, as in transmit operations, or it may be used as an extension of the first address, specifying field length, byte size and accumulator offset for variable field length operations. In any event, both address parts are independently indexable.





Index Format

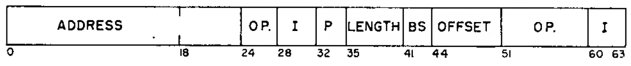
The format of an index word consists of the value, count and refill fields and the index flag bit. The value field is leftmost in the index. It contains 25 bits that are interpreted as an address of 24 bits and a sign bit. The count and refill fields each contain 18 bits and are adjacent to each other in the rightmost part of the index. The index flag bit is adjacent to the value sign bit. The format, as shown below, has two bits between the index flag and the count field which are not used in indexing operations.



In input-output operations, the index word is used to control data transmission to and from the exchange. In these operations the index flag and the two bits between the index flag and the count field are used as control bits. Part of the value field is used for status bits. The definition of these bits allows an index word to be used both for input-output and indexing operations.

Operand Address

The format of the operand is determined by the operation specified in the instruction. A full word operand is implied by floating point operations. These operations have an address part of 18 bits. A half word operand is implied by most branch, index modification, and transmit operations. This group of operations has an address part of 19 bits. The 18 high-order bits address full words. The low-order bit addresses the first or second half of the full word. A variable field operand is implied by integer arithmetic and connect operations. This group of operations, the integer arithmetic and connect operations, has an address part of 24 bits. The 18 high-order bits address full words. The six low-order bits address the bit in the addressed word which corresponds to the left end of the field. The variable field length operations are specified in a full-length instruction. The length of the field is specified in bits 35-40 of the instruction and completes the definition of the operand location. Zero in bits 35-40 indicates a 64-bit field.



The address part of the instruction may be modified by indexing as described in the next section. The final address thus obtained is called the effective address. When no indexing occurs, the address part of the instruction, extended with the necessary low-order zeros, is the effective address.

There are three modes of addressing. The most frequently used mode is direct addressing. In this mode, the effective address is used to specify the storage location of the operand. A second mode of addressing is immediate addressing. In this mode, the effective address itself constitutes the operand. Up to 24 bits of information may be available in the effective address. Zeros are supplied to the right of the effective operand field when a field length of more than 24 is specified. The sign of the effective address is ignored. Immediate addressing is available for some operations. A third mode of addressing is indirect addressing. In this mode, the effective address is used to obtain a second level effective address, which subsequently can be used for addressing the operand. The indirect mode of addressing is made possible by the operation LOAD VALUE EFFECTIVE. The immediate, direct, and indirect mode of addressing, taken in this order, require an increasing number of storage references. The exact number of storage references for each mode depends upon the amount of indexing which is specified.

Besides the specified operand, a second operand is required in most operations. In the majority of operations, the second operand is implied. In floating point arithmetic, the implied operand is in the accumulator unless otherwise stated. In branch operations, the implied operand may be considered to be in the instruction counter. In index arithmetic, the second operand must be stated specifically by the second index field. In transmit operations, the second operand is stated explicitly in the second half of the instruction. In integer arithmetic and connect operations, the contents of the accumulator is implied as the second operand. The offset field is used to specify the number of bits between the low-order bit of the field and the low-order end of the accumulator. An offset of zero indicates that the field lines up at the right end of the accumulator.

PROGRAMMING NOTE

In the section "Address Insertion," the operation STORE VALUE IN ADDRESS is described. This operation permits convenient insertion of an address part of proper length in an instruction.

Indexing

In most half-length instructions, an index address field, I, is available for indexing operations. In full-length instructions, two fields I are available, one in each half-word. The field I normally contains four bits. The index address, when zero, normally indicates that no indexing is to take place. When the index address is not zero, it indicates which of the 15 available index registers X1-X15 is to be used in the indexing operation. Each index register occupies one full-word storage location. The registers X1-X15 have addresses 17-31. The use of a storage location for an index register does not preclude its being addressed and used in the standard fashion. In conditional branch instructions, the indexing choice is limited between 0 (no indexing) and 1 (use index register X1). In this class of instructions, the index field is only one bit.

When an index register is specified in the index field I of an instruction, its contents may be used for address modification or for progressive indexing. The modification of the address part of the instruction is possible in almost all instructions. One group of instructions do not permit address modification. In these instructions, the index address field I is not available. Progressive indexing is possible only for the variable-field-length operations.

Address Modification

In address modification, the value part of the index is added to the address part of the instruction in order to obtain the effective address. The addition is algebraic. The value field of the index is interpreted as a signed 24-bit field. The address part of the instruction is interpreted as a positive unsigned field. The length of the address field depends upon the instruction class. The missing low-order bits, if any, are always assumed to be zero. In the addition of index and instruction address, an overflow bit may be produced. This overflow bit, which is a carry produced in the high-order bit position, is ignored. The overflow bit may occur when a complement notation is used to obtain the equivalent of negative instruction addresses.

The index and the address part of an instruction half-word normally are used together. In the variable field instructions, the 19 bits which form the length, byte size, and offset field are indexed as a single field. Carries between parts of this field, if present, are not suppressed.

Index register X0, storage location 16, is not available for address modification. An I field value of zero indicates that no modification is to be applied.

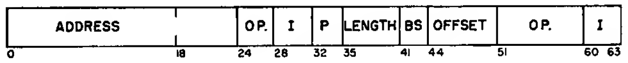
The effective address which results from an indexing operation always has 24 numeric bits and a sign. Depending upon the operation, a full-word, half-word, or bit-address may be required. In each case, only the applicable part of the effective address is used. The non-applicable part is ignored. The effective address sign is not used in storage addressing or immediate data fields.

When an effective word address is obtained which does not correspond to a core storage location that is physically available in a particular installation, indicator AD, address invalid, is turned on. An instruction with an invalid data address is executed as a NO OPERATION instruction. A branch instruction with an invalid address is also executed as a NO OPERATION. A negative effective address is not considered to be invalid; its magnitude is used.

The index flag, index result, and index comparison indicators, XF, XCZ, XVLZ, XVZ, XVGZ, XL, XE, and XH, are not changed when an index is used in address modification.

Progressive Indexing

In variable-field-length instructions, the field P, bits 32-34, is available to specify the choice between address modification and progressive indexing. The codes 0 and 4 in the field P indicate standard address modification. The codes 1-3 and 5-7 indicate progressive indexing.



The codes 0 and 4, which both specify standard address modification, indicate a difference in the mode of addressing. For code 0, the direct mode is specified and the effective address is used to address the operand. For code 4, the immediate mode is specified and the effective address is used as the operand itself. This mode is applicable to all operations of the fetch type in the connect and integer class. Operations of the store type are not possible with immediate addressing, since no storage location is specified. When a store type operation is given with immediate addressing, the instruction is executed as a NO OPERATION and indicator OP (operation code invalid) is turned on. For operation classes other than variable-field-length instructions, immediate addressing is specified for the individual operations in which it is available.

In progressive indexing, the value field, bits 0-24 of the index specified by I, is used as the effective address. If progressive indexing with index register X0 is specified, indicator OP (operation code invalid) is set. Subsequent to this use as the effective address, the address part, bits 0-23, of the instruction is added to the value field. The address part is used as a positive unsigned quantity. The addition is algebraic and takes into account the sign, bit 24, of the index value.

For code 1, the addition takes place as defined above. For code 2, addition takes place and also the index word is counted down. For code 3, addition and counting take place and also the index specified by I is refilled if the count reaches zero.

For codes 5-7, the operations are identical to those for codes 1-3, with the exception that the address part of the instruction is algebraically subtracted rather than added. The operations then are: 5 subtract from value; 6 subtract from value and count; and 7 subtract from value, count and refill.

Progressive indexing combines immediate index incrementing with any one of the variable field length operations. It is possible only when the effective address can be specified in an index quantity and requires no further address modification.

The index flag and index result indicators are set following each progressive index operation. They are set according to the new contents of the index register specified by I. The setting occurs prior to the refill operation which may be specified.

The index field I controlling progressive indexing is that located in the first half of the full-length instruction. The index field I in the second half of the instruction may be used to modify bits 35-50 of the instruction. On doing this, bit position 3 of the index word is aligned with bit position 35 of the instruction; the three high-order bits of the index word are ignored. If the same index register is specified in both halves of an instruction while using progressive indexing, the value of the index register before modification will be used in indexing the second half of the instruction.

Programming Notes

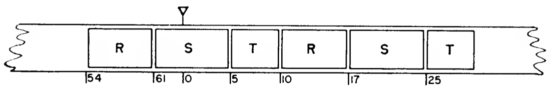
An operation in which progressive indexing is specified is executed in two parts. The variable field length part of the operation is prepared and then the index modification portion of the operation is performed. If an exception condition is detected during the variable field length portion of the operation, the entire operation will be suppressed, and the appropriate indicator will be turned on. However, if an exception condition is detected during the index modification portion of the operation, only this portion will be suppressed, the variable field length portion having been executed already.

If the index register specified by the I field of an operation using progressive indexing is an operand in the variable field length portion of that operation, the original value of this index register will be used.

A CONNECT operation with connective 0101, with a progressive indexing operation, can be used as an immediate increment operation without disturbing the accumulator. The progressive indexing mode of operation is the only means for obtaining an immediate increment of 24 bits.

Programming Example

Stored in core storage starting at location LIST is a list of ten data records. Each record consists of three fields, R, S, and T, each an unsigned binary integer. Field R is seven bits long, field S is eight bits long,



and field T is five bits long. For each record add field T to field R and place the result in field S. Figure 7 illustrates the use of progressive indexing.

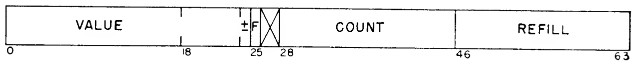
NAME	STATEMENT	NOTES
	LX, $X12, CTLWD	1
LOOP	L(V+I) (BU, 7, 8), 0.15 ($X12), 0	2
	+(V-I)(BU, 5, 8), 0.08 ($X12), 0	3
	ST(V+ICR)(BU, 8, 8), 0.13($X12), 0	4
	BZXCZ, LOOP	5
CTLWD	XW, LIST, 10, CTLWD	

Notes:
1. Loads index register from control word.
2. Loads field R into the accumulator, increments index register to address field T.
3. Adds field T, increments index register back to address field S.
4. Stores result in field S, increments index register to address field R of next record, counts down index, and tests count. If count is zero, refills index to its original value, thus permitting repetition of program to start at LOOP.
5. Test for last record in list.

Figure 7. Progressive Indexing

Index Arithmetic

The three fields of an index word are intended for three distinct purposes. The value field contains the quantity which is combined with the instruction address in order to obtain the effective operand address in instruction address modification. The value can be changed by increment operations. The count field registers the number of times an increment is to be applied. The contents of the count field are reduced by one whenever a count is specified. The count field contains no sign. On reaching zero, the count proceeds to 262,143 ($2^{18} - 1$). The refill field contains the address of a new index word that can replace the original index word. A refill operation will occur only if it is specified by an instruction. The refill can be made conditional on the count's reaching zero, or it can be specified to occur unconditionally. A refill address of zero will replace the index word with all zeros. The instruction set takes full advantage of the use of the index fields for the above purposes. However, a different use of the index fields is possible if desired, and sufficient operations are available to permit flexibility in this respect.



The set of operations provided for index arithmetic can be divided into seven groups: direct index arithmetic, immediate index arithmetic, refill operations, count and branch operations, named index loading, multiple index loading and indirect index loading.

Direct index arithmetic permits loading, storing, incrementing, and comparing of index quantities. The operand is specified in the instruction by the effective address. The effective address may be obtained by modifying the instruction address part with an index quantity. The operand is obtained from storage or stored in storage.

Immediate index arithmetic permits loading, incrementing, and comparing of index quantities. The operand is contained in the instruction itself and not subject to further modification.

The count and branch instructions combine an index arithmetic operation with a conditional branch operation. The branch is conditioned by the result of an index count. The indexing operation is a special case of the immediate increment operation.

The remaining groups of index arithmetic permit loading of an index register only.

The index word to be modified by index arithmetic is specified in the field marked J, instruction bits 19-22. When one of the numbers 1-15 is specified in field

J, the corresponding index word X1-X15 will be used in the operation. These words are in storage locations 17-31. When the field J is 0, storage location 16 is used in the operation. This location is named X0.

PROGRAMMING NOTE
X0 cannot be used directly for address modification, since a zero I field in such an instruction specifies no indexing. However, if progressive indexing is specified, a zero I field will cause the indicator OP (operation code invalid) to be set. X0 may take part in any index arithmetic instruction, since a zero J field always refers to location 16, X0.

Index Indicators

Eight indicators are used to describe the result of index arithmetic. These are the index flag indicator XF, the index result indicators XCZ, XVLZ, XVZ, and XVGZ, and the index comparison indicators XL, XE, and XH.

The index flag indicator is set in any index operation that specifies an index in the field J. These include index arithmetic and count and branch operations. This indicator is further set in progressive indexing operations, in which case the index is specified in the I field of the left half of the instruction. Finally, this indicator is set in the refill operations according to the index specified in the address field of the instruction. The index result indicators are set in all cases in which the index flag indicator is set, except the index comparison operations. The indicators are not set when an index is used for address modification or to specify the count in a transmit operation. Also, the indicators are not set when index quantities are modified by other operations such as variable field length or transmit operations.

The indicators describe the index after the index modification is completed, but before any refill operation conditioned by the modification takes place. In the operations REFILL and REFILL ON COUNT ZERO, also, the indicators describe the index before the refill operation takes place. The particular conditions applying to each indicator are summarized below.

Index Flag (XF). This indicator is set to one when the index flag bit, bit 25, is one. It is set to zero when the index flag bit is zero.

Index Count Zero (XCZ). This indicator is set to one when the count field is zero. It is set to zero when the count field is not zero.

Index Value Less than Zero (XVLZ). This indicator is set to one when the value field is non-zero and negative. It is set to zero when the value field is zero or positive.

Index Value Zero (XVZ). This indicator is set to one when the value field is zero. It is set to zero when the value field is not zero.

Index Value Greater than Zero (XVGZ). This indicator is set to one when the value field is non-zero and positive. It is set to zero when the value field is zero or negative.

The index comparison operations compare the value or the count field with a quantity specified by direct or immediate addressing. The index comparison indicators XL, XE, and XH described below are set for the comparison operations instead of the index result indicators. The names of the index comparison indicators refer to the index specified in the J field of the instruction as it is compared with the quantity specified by the operand.

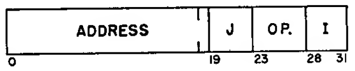
Index Low (XL). This indicator is set to one when the field in the index specified by J is lower than the comparand. It is set to zero when the index field is equal to or higher than the comparand.

Index Equal (XE). This indicator is set to one when the field in the index specified by J is equal to the comparand. It is set to zero when the index field is lower or higher than the comparand.

Index High (XH). This indicator is set to one when the field in the index specified by J is higher than the comparand. It is set to zero when the index field is equal to or lower than the comparand.

Direct Index Arithmetic

The direct index arithmetic operations make use of the format shown below. The format includes two index fields. The rightmost field, marked I, is used in standard fashion to produce an effective operand address. In the majority of these operations, 19 bits of the effective address are used in order to permit addressing of half-words. The two exceptions use only 18 bits of the effective address because they require full-word operands. The index field J specifies the index register upon which the operation is performed.



Load Index (LX)

The index word specified by J is replaced by the full-word specified by bits 0-17 of the effective address.

Load Value (LV)

The value field, bits 0-24, of the index word specified by J is replaced by bits 0-24 of the half-word specified by bits 0-18 of the effective address.

Load Count (LC)

The count field, bits 28-45, of the index word specified by J is replaced by bits 0-17 of the half-word specified by bits 0-18 of the effective address.

Load Refill (LR)

The refill field, bits 46-63, of the index word specified by J is replaced by bits 0-17 of the half-word specified by bits 0-18 of the effective address.

Store Index (SX)

The index word specified by J is stored in the full-word memory location specified by bits 0-17 of the effective address.

Store Value (SV)

The value field, bits 0-24, of the index word specified by J replaces bits 0-24 of the half-word specified by bits 0-18 of the effective address.

Store Count (SC)

The count field, bits 28-45, of the index word specified by J replaces bits 0-17 of the half word specified by bits 0-18 of the effective address. Bits 18-24 of the half word are set to zero.

Store Refill (SR)

The refill field, bits 46-63, of the index word specified by J replaces bits 0-17 of the half word specified by bits 0-18 of the effective address. Bits 18-24 of the half word are set to zero.

Add to Value (V+)

The address part, bits 0-24, of the half-word addressed by bits 0-18 of the effective address is added to the address part, bits 0-24, of the index word specified by J. Addition is algebraic, taking into account the sign, bit 24, of both addresses.

Add to Value and Count (V+C)

This operation is identical to ADD TO VALUE, except that in addition the count field of the index word specified by J is counted down by one.

Add to Value, Count and Refill (V+CR)

This operation is identical to ADD TO VALUE AND COUNT, except that in addition the index word specified by J is refilled if the count reaches zero as a result of this operation.

PROGRAMMING NOTES

The refill does not occur if the count was zero prior to the operation, since in that case the count reaches the value $2^{18} - 1$ as a result of this operation.

The index-flag and index-result indicators are set in all of the preceding direct index arithmetic operations. The index-comparison indicators are unchanged.

Compare Value (KV)

The address part, bits 0-24, of the half-word addressed by bits 0-18 of the effective address is compared with the address part, bits 0-24, of the index word specified by J. Comparison is algebraic, taking into account the sign, bit 24, of both addresses. Positive and negative zero are considered equal.

Compare Count (KC)

Bits 0-17 of the half-word specified by bits 0-18 of the effective address are compared with the count field, bits 28-45, of the index word specified by J. The comparison is unsigned and for magnitude only.

The index-flag and index-comparison indicators are set in each of the last two operations; the index result indicators are not set in these operations.

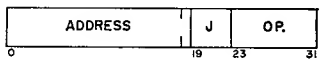
PROGRAMMING NOTES

The LOAD INDEX and STORE INDEX operations involve transmission of full words. In all other operations the effective operand address refers to a half-word. Of this half-word, either the first 18 or the first 25 bits are used, depending on the operation to be performed. The remaining bits are ignored and unchanged. When instructions or data are used as the operand in increment and compare operations, proper care should be taken that the left 25 bits are usable as a signed quantity. Note further that the first 25 bits of the half-word have been numbered 0-24. However, the actual bit address of this field may be either 0 or 32.

The overflow which may occur in index arithmetic is ignored.

Immediate Index Arithmetic

The immediate index arithmetic operations make use of the format shown below. The index field J contains the address of the index to which the operation applies. The address part contains the operand. A 19-bit operand can be specified. In some operations, only 18 of these 19 bits are used. In this class of operations, the index field I is not available and no address modification by indexing takes place.



Load Value Immediate (LVI)

Bits 0-18 of the value field of the index word specified by J are replaced by the address part, bits 0-18, of the instruction. The remaining bits of the value field, bits 19-24, are set to zero. In so doing, the sign of the address contained in the value field is made positive.

Load Count Immediate (LCI)

The count field, bits 28-45, of the index word specified by J is replaced by bits 0-17 of the address part of the instruction.

Load Refill Immediate (LRI)

The refill field, bits 46-63, of the index word specified by J is replaced by bits 0-17 of the address part of the instruction.

Load Value Negative Immediate (LVNI)

Bits 0-18 of the value field of the index word specified by J are replaced by the address part, bits 0-18, of the instruction. Bits 19-23 of the value field are set to zero. The sign bit, bit 24, of the value field is set to one. In so doing, the sign of the address contained in the value field is made negative.

Add Immediate to Value (V+I)

The address part, bits 0-18, of the instruction is added to bits 0-18 of the value field of the index word specified by J. Zeros are added to bits 19-24 of the value field. Addition is algebraic, taking into account the sign, bit 24, of the value field.

Add Immediate to Value and Count (V+IC)

This operation is identical to ADD IMMEDIATE TO VALUE except that in addition the count field of the index word specified by J is counted down by one.

Add Immediate to Value Count and Refill (V+ICR)

This operation is identical to ADD IMMEDIATE TO VALUE AND COUNT except that in addition the index specified by J is refilled if the count reached zero.

Subtract Immediate from Value (V—I)

Subtract Immediate from Value and Count (V—IC)

Subtract Immediate from Value, Count and Refill (V—ICR)

These three operations are identical to the preceding three ADD IMMEDIATE TO VALUE operations, except that the address part of the instruction is subtracted from, not added to, the value field of the index word specified by J.

Add Immediate to Count (C+I)

Bits 0-17 of the address part of the instruction are added to the count field, bits 28-45, of the index word specified by J. Both quantities are unsigned and the addition takes place modulo 2^{18}.

Subtract Immediate from Count (C—I)

Bits 0-17 of the address part of the instruction are subtracted from the count field, bits 28-45, of the index word specified by J. Both quantities are unsigned and the subtraction takes place modulo 2^{18}.

The index flag and index result indicators are set in all of the preceding immediate index arithmetic operations. The index comparison indicators are not changed.

Compare Value Immediate (KVI)

The address part, bits 0-18, of the instruction is compared with bits 0-18 of the value field of the index word specified by J. Zeros are compared with bits 19-24 of the value field. Comparison is algebraic, taking into account the sign, bit 24, of the value field. Positive and negative zero are considered equal.

Compare Value Negative Immediate (KVNI)

The address part, bits 0-18, of the instruction is compared with bits 0-18 of the value field of the index word specified by J. Zeros are compared with bits 19-23 of the value field. A one, or negative sign, is compared with the sign, bit 24, of the value field.

Compare Count Immediate (KCI)

Bits 0-17 of the address part of the instruction are compared with the count field, bits 28-45, of the index word specified by J. The comparison is unsigned and for magnitude only.

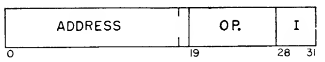
The index flag and index comparison indicators are set in each of the last three operations; the index result indicators are not set in these operations.

PROGRAMMING NOTE

To set the index flag and index result indicators according to the status of an index without modifying that index, an ADD IMMEDIATE TO VALUE operation with zero address part can be used as one of several possibilities.

Refill Operations

The refill operations make use of the format shown below. These operations make possible the refilling of any index word in storage. The index word specified by the I field is used in standard fashion to form an effective address. Eighteen bits of the effective address are used to specify the full word in storage that is to be refilled. The refill operation can be made conditional (the count of the addressed index word to be zero) or it can be specified to occur unconditionally. A refill operation replaces an index word with the word addressed by the refill field of the original index word.



Refill (R)

The full word addressed by bits 0-17 of the effective address is replaced with the full word that is addressed by the refill field, bits 46-63, of the word at the effective address.

The index flag and index result indicators are set according to the original contents of the word at the effective address.

Refill on Count Zero (RCZ)

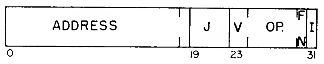
This operation is the same as REFILL, except that the refill will occur only if the count field, bits 28-45, of the addressed word is zero.

PROGRAMMING NOTE

The index-count-zero indicator may be used to find out whether the refill operation took place in REFILL ON COUNT ZERO.

Count and Branch Operations

The count and branch operations make use of the format shown below. The format includes two index fields. The rightmost field, marked I, is used in standard fashion to produce an effective operand address. The I field is a one-bit field and provides a choice between no address modification and the use of X1 for address modification. The other index field, marked J, designates an index register to be counted down. After this count operation, the index flag and index result indicators are set according to the resulting contents of index register J. The resulting count is then tested in a conditional branch operation. The condition for a successful branch is specified by instruction bit 30. If this bit is one, the branch is performed if the resulting count field of index register J is zero. If this bit is zero, the branch is performed if the resulting count field is non-zero. The effective address of the instruction is used as the branch address.



The count and branch operations may be combined with the immediate incrementing of the value field by 1, ½, or –1, as specified by the code in instruction bits 23 and 24. According to this code, a one may be added in bit position 17 or 18, or subtracted in bit position 17, of the value field of the index word specified by J. The addition or subtraction is algebraic and takes into account the sign, bit 24 of the value field. This addition or subtraction is termed advancing. The four codes are:

00	Leave value unchanged.
01	Add half to value.
10	Add one to value.
11	Subtract one from value.

The increment, count, and refill operations are independent of the success of the branch. The index result indicators are set after the value field is incremented. The setting of the index flag and index result indicators precedes the refill operation. If the index flag indicator is turned on, if its mask bit is one, and if the interruption system is enabled, the entire index branching operation except the actuation of this indicator is suppressed. If fields I and J both specify the same index register, the effective instruction address is formed before any modification of the contents of the index register.

Count and Branch (CB)

The index word specified by J is counted down. The value field of the index is incremented as specified. Subsequently a conditional branch is performed conditioned by the value of the resulting count field in J.

Count, Branch and Refill (CBR)

This operation is identical to COUNT AND BRANCH, except that in addition the index word specified by J is refilled if the count reached zero.

PROGRAMMING EXAMPLE

It is required to obtain the scalar product of two vectors, A and B. Each vector has n elements, each a single precision floating point word. Vector A has its first element at a_0; vector B has its first element at b_0.

The product is to be stored at c_0. A is stored in successive core storage locations. B is a column vector of a matrix whose rows have p elements and are stored in successive storage locations. Therefore, the elements of B have locations which are p words apart. Figure 8 shows the program.

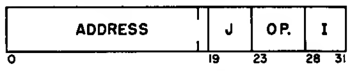
NAME	STATEMENT	NOTES
	LX, \$X7, CTLWD	1
	LX, \$X8, CTLWD	1
START	L(N), TRUZERO	2
LOOP	LFT(N), a_0 (\$X7)	3
	*+(N), b_0 (\$X8)	4
	V+ICR, \$X8, p.0	5
	CBR+, \$X7, LOOP	6
	ST(N), c_0	
TRUZERO	DD(U),0E-600	
CTLWD	XW, 0.0, n, CTLWD	

Notes:
1. Set index registers to initial values. These instructions are necessary only when the program is first loaded.
2. Resets accumulator to true zero. Successive executions of the program may start at this point. The XPFN indicator is turned on as the result of the range of the floating point number loaded into the accumulator.
3. Loads element of A in factor register.
4. Multiplies by element of B, adds product to accumulator.
5. Increments to next element in B, and counts down the index. If a zero count is reached, the index register is refilled - ready for the next execution of the program.
6. Increments to the next element in A and counts down the index. Branches back to the start of the loop unless the last element is reached, in which case the index is refilled and the result stored.

Figure 8. Example of Count, Branch and Refill

Named Index Loading

Fifteen index registers are directly available to the programmer for use in address modification. In some applications it may be desirable to use a larger number of index registers. Such an application may require frequent loading and storing of index words. In order to simplify the loading and storing of index words, the method of named index loading has been provided. When this method of index loading is used, X0 contains the core storage address, or name, of the index word last loaded into one of the registers X1-X15 by a named index load. When a named index load is specified, the contents of the index register are first stored in the core storage location specified by X0. The name of the new index word is then placed in X0 and the word itself placed in the specified register. The named index load operation, RENAME, makes use of the format shown below. The field J specifies the index register which is to be stored and loaded. The field I is used in standard fashion to form an effective address.



The RENAME instruction makes it possible to use any core storage word conveniently as an index quantity.

Rename (RNX)

The index word in the index register specified by the field J is stored in the location addressed by the refill field of X0. The refill field of X0 is then replaced by bits 0-17 of the effective address. The remaining bits of X0 remain unchanged. Next, the index word specified by J is replaced by the word in core storage addressed by the new refill field of X0. When either the old or the new name of the index refers to one of the locations 0-31, the instruction is executed as no-operation and indicator AD is set. If the interrupt system is enabled and if the old name is both out of bounds and invalid, the AD and DS indicators are set. If the new name is both out of bounds and invalid, the AD and DF indicators are set. The index flag and index result indicators are set according to the new contents of the index register specified by J.

PROGRAMMING NOTE

In many applications the RENAME instruction can be considered as a prefix to another operation which expands the index address field I of that operation to include any address in core storage.

PROGRAMMING EXAMPLE

One hundred words of floating point data are placed in a block starting at location DATA. Each word is to be stored in one of eight blocks, A through H, according to the binary number, from 0 to 7, formed by the three flag bits in its sign byte. The control words defining the position and length of each of these eight blocks are stored in order in successive locations starting at location CTLWD1. Assume that index registers X14 and X15 and the refill field of X0 are available to the program, but that all the remaining index registers are occupied. Also assume that the data flag indicators are masked off. See Figure 9.

Multiple Indexing

In some applications index quantities are derived as the sum of a selected number of basic quantities. As the basic quantities are modified, the derived quantities should change accordingly. It may be convenient

NAME	STATEMENT	NOTES
	LRI, $X0, 0.0	1
	RNX, $X14, CTLWD0	2
LOOP	L(BU, 3,8), 0.61 ($X14), 14	3
	LV, $X15, 9.32	4
	LWF (U), 0.0($X14)	5
	RNX, $X14, CTLWD1($X15)	6
	BXCZ, ERROR	7
	ST (U), 0.0($X14)	8
	V+IC, $X14, 1.0	9
	RNX, $X14, CTLWD0	2
	CB+, $X14, LOOP	10
ERROR	Start of error routine for too many words in block	
CTLWD0	XW, DATA, 100	
CTLWD1	XW, A, n_a	
CTLWD8	XW, H, n_h	

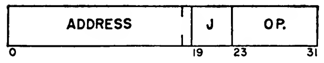
Notes:
1. Clears refill field of X0.
2. Sets up index for DATA.
3. Loads flags in accumulator.
4. Places flags in word address portion of X15.
5. Loads floating point data.
6. Places control word for proper block in X14.
7. Branches to error routine if block is full.
8. Stores floating point data.
9. Increments and counts control word for block.
10. Increments and counts control word for DATA. Tests for last word.

Figure 9. Rename Program Example

to update only the basic quantities and form a derived sum when that particular sum is required. In order to facilitate this mode of operation, it should be possible to specify in one instruction a group of index registers and indicate which registers of the group should be used in forming the derived sum. This mode of indexing is called multiple indexing. The index registers which participate in multiple indexing are named X0-X15. They are the words in storage locations 16-31.

In multiple indexing, bits 0-15 of the address part of the instruction are used to indicate which index registers participate in the operation. When bit position n contains a one, the register Xn participates. When bit position n contains a zero, the register Xn does not participate. Bits 16-19 of the address part of the instruction are ignored.

The multiple indexing operation makes use of the instruction format shown below. The index field J contains the address of the index to which the operation LOAD VALUE WITH SUM applies. No address modification is available for this operation.



Load Value with Sum (LVS)

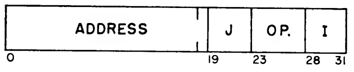
The address parts, bits 0-24, of the index registers which participate are added and replace bits 0-24 of the index word specified by J. The addition is algebraic, taking into account the sign, bit 24, of all quantities. Bits 25-63 of the index word specified by J remain unchanged. If bits 0-15 of the address part of the instruction are all zero, the value field of the index register specified by J is cleared.

The index flag and index result indicators are set according to the new contents of the index register specified by J.

Indirect Addressing

The effective address of an instruction is normally used to address the data upon which the instruction operation is performed. In another mode of addressing, the effective address can be used to address another instruction. This instruction, in turn, contains an operand address and index which can be used to form a second level effective address. In some applications, it is desirable to address the operand by the second level effective address rather than by the effective address of the original instruction. The process of using a first level effective address to obtain a second level effective address is called indirect addressing. The second level effective address, of course, can refer to another instruction of which, again, the effective address can be obtained. In this manner it is possible to extend indirect addressing through many levels.

Indirect addressing is made possible by the instruction LOAD VALUE EFFECTIVE whose format is shown below.



Load Value Effective (LVE)

Bits 0-24 of the index word specified by J are replaced by the effective value plus sign of the instruction found at the location specified by the effective address. When the instruction found at the location specified by the effective address is another LOAD VALUE EFFECTIVE instruction, the process is repeated with that instruction. The final effective value plus sign is placed in the index register specified by the field J of the original instruction.

All instructions encountered in a multiple level indirect addressing process, except for the last one, are LOAD VALUE EFFECTIVE instructions. The instruction which produces the final effective address is any instruction other than LOAD VALUE EFFECTIVE. It is assumed to be either a half length instruction or the first half of a full length instruction. The final effective address is produced according to the rules which apply to that instruction. In particular, the lengths of the address and index fields are determined from the instruction class to which the instruction belongs. No check is made for invalid operation codes.

Although the address part of the second half word of TRANSMIT and SWAP instructions is 19 bits, only the first 18 bits of the address field are used when the second half of those instructions is encountered in the process. If the second half word of any other type of full length instruction is encountered, the full 19 bits of the second address are used.

When the first half of any variable field length instruction terminates the operation, the address modification mode of indexing is assumed, even though the full instruction may use the progressive indexing mode.

Even though the instruction class of the final instruction is determined in order to obtain the proper effective address, the instruction operation is not executed. The operation LOAD VALUE EFFECTIVE, which extends the indirect addressing to another level, is always executed. The index register in which the final effective address is to be stored is specified by the index field J of the original LOAD VALUE EFFECTIVE instruction. The fields J of all subsequent instructions encountered in the operation are ignored.

In indirect addressing, it is possible that addresses refer to each other in such a way that they form a loop. As a result, the computer cannot finish its operation and no interruption is accepted. This situation would be the result of a programmer's error. In order to prevent the machine from being locked up in this way, the LOAD VALUE EFFECTIVE operation will be terminated, if still active, after one millisecond and indicator USA, unended sequence of addresses, will be turned on. At time of termination, the final effective address is placed in the index register specified by the J field of the original instruction.

Index flag and index result indicators are set according to the new contents of the index specified by J.

PROGRAMMING EXAMPLE

Suppose that a floating point result exception has occurred. The instruction counter was stored in location IC. As part of the interruption correction routine, it is necessary to obtain the operand of the floating point operation that produced the exception, placing this operand in the accumulator. Assume that the operand is single precision. The instructions shown in Figure 10 accomplish this.

NAME	STATEMENT	NOTES
	LX, $X1, IC	1
	LVE, $X15, -0.32 ($X1)	2
	L (U), 0.0 ($X15)	3

Notes:
1. Loads instruction counter in index register X1.
2. Places effective address of previous instruction in X15. Since this instruction is a floating point operation, it is located a half word back from the stored instruction counter reading. The negative address is obtained by using the two's complement.
3. Loads operand of floating point instruction into accumulator.

Figure 10. Indirect Addressing Example

Address Insertion

The length of the address field of an instruction may be 18, 19, or 24 bits, depending upon the operation class of the instruction. In order to facilitate the insertion of an address of proper length in an instruction, the operation STORE VALUE IN ADDRESS is provided. This operation replaces the address field of an instruction with a field of proper length. The field is obtained from the value field of an index word. The instruction addresses a half word and assumes it to be either a half length instruction or the first half of a full length instruction. The length of the address field is determined from the instruction operation code.

The STORE VALUE IN ADDRESS operation makes use of the format shown below. The index field J specifies the index register from which the address is to be obtained. The location of the instruction in which the address will be inserted is given by the effective address of the instruction.

ADDRESS	J	OP.	I
0	19	23	28 31

Store Value in Address (SVA)

The first 18, 19, or 24 bits of the value field of the index word specified by J replace the address field of the instruction specified by the effective address.

The index flag and index result indicators are set according to the contents of the index register specified by J.

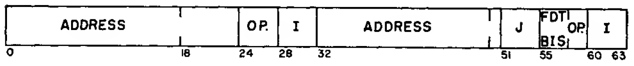
PROGRAMMING NOTE

Although the address part of the second half-word of TRANSMIT and SWAP instructions is 19 bits, only the first 18 bits of the address field are replaced when the second half-word is referred to by the STORE VALUE IN ADDRESS instruction. If an address is stored in the second half-word of any other type of full-length instruction, the full 19 bits of the second address are replaced.

Data Transmission

A set of instructions is available to transmit data from one storage location to another. One word or a group of words may be transmitted in a single operation. The number of words to be transmitted is specified by a count. The count may be immediate, or it may be contained in an index word. Up to $2^{18} - 1$ (262,143) full words may be transmitted in one operation.

Transmission may be unidirectional or bidirectional. In unidirectional transmission, data are moved from one storage location to another storage location. After completion of the operation, the data remain unchanged in the original storage locations. The data which were originally in the receiving storage locations are destroyed. In bidirectional transmission, data are interchanged. Following this operation, both sets of data are preserved, but they have interchanged their locations. Unidirectional transmission of full words is specified by the TRANSMIT operation. Bidirectional transmission of full words is specified by the SWAP operation.



A full-length instruction is used for transmit operations. The format contains two addresses. Each address can be modified by the index word specified by the I field in its respective half-word. The two effective addresses obtained as a result of the modification specify the starting addresses of the two storage areas which participate in the word transmission. Only 18 bits of each effective address are used.

Bits 55-57 of the data transmission instructions are used as modifiers. The choice between unidirectional and bidirectional transmission is specified by modifier bit 57. A zero specifies unidirectional transmission; a one specifies bidirectional transmission. The choice between an immediate or a direct count is specified by modifier bit 56. A zero specifies a direct count; a one specifies an immediate count. The choice between forward and backward transmission is specified by modifier bit 55. A zero specifies forward transmission; a one specifies backward transmission.

In direct count operations, the field J, bits 51-54, specifies an index register. The count field, bits 28-45, of the index word in this register is used as a full word count. A count field of zero is interpreted as 2^{18}. Since this many words cannot be transmitted without encountering an invalid address, such a count is never valid. It is not detected, however, until the data transmission actually encounters an invalid or protected address. The count field in the index register is not changed as a result of the operation, and the index result indicators are not affected. A zero J field refers to location 16, X0.

In immediate count operations, the full word count is specified by instruction bits 51-54. A zero count field indicates the maximum count of 16 full words.

In forward transmission, the sending and receiving addresses of each successive word transmitted are obtained by incrementing the previous addresses by one. The starting addresses specified in the instruction are therefore the addresses of the first word in each of the two storage areas.

In backward transmission, the sending and receiving addresses of each successive word transmitted are obtained by decrementing the previous addresses by one. The starting addresses specified in the instruction are therefore the addresses of the last word in each of the two storage areas.

Transmit (T)

The number of full words specified by the count is transmitted from the area which starts at the location addressed by bits 0-17 of the left effective address to the area which starts at the location specified by bits 0-17 of the right effective address.

Swap (SWAP)

The number of full words specified by the count is transmitted from the area which starts at the location addressed by bits 0-17 of the left effective address to the area which starts at the location specified by bits 0-17 of the right effective address. At the same time, the full words of the area which starts at the right effective address are transmitted to the area which starts at the left effective address.

Programming Notes

The difference between backward and forward transmission is important. When the receiving area overlaps the upper end of the sending area, backward transmission must be used; if the receiving area overlaps the lower end of the sending area, forward transmission must be used.

All full-length instructions are assumed to have a 24-bit address part in the left half-word. Since data transmission instructions are full length, 24 bits of the left half-word of the instruction are used in forming the effective address. However, only 18 bits of the effective address are used by these operations.

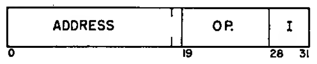
Programming Example

Storage locations 112, 113, and 114 contain the floating point constants a, b, and c. It is desired to rotate these constants so that their order is b, c, and a. The instruction which accomplishes this rotation is:

SWAPI, 2, 112.0, 113.0

Data Reset

Several instructions are available which set the bits in a storage field to zero. The value, count, and refill fields of an index word can be set to zero with an immediate load instruction. With the CONNECT TO MEMORY operation, using connective 0000, a variable storage field may be set to zero. In order to facilitate setting all the bits in a full storage word to zero, the instruction STORE ZERO is provided. This is a half-length instruction which has the format shown below. When the instruction is followed by a COUNT AND BRANCH operation in a programmed loop, a storage area of several full words may be set to zero.



Store Zero (Z)

An all-zero word is stored in the full storage word location specified by bits 0-17 of the effective address.

Address Monitoring

In multiprogrammed operation, when several programs are in core storage at one time and are executed in intermixed fashion, it is important that errors in one program be prevented from causing changes in data or instructions belonging to another program. This protection may be provided by monitoring all addresses at which data are to be stored and suppressing the storage if an address lies within the protected area of core storage, as defined by the contents of two boundary registers.

By extending the address monitoring to include addresses from which instructions and data are fetched, it may also be used to verify the proper execution of a program and to detect incorrect instructions. For instance, in program check-out, incorrect instructions or data addresses, especially those which are the result of address computation, may be detected by this means.

The monitoring procedure can be refined by changing boundaries as the program proceeds. This refinement can be used to ascertain that the addresses used in each program section do not belong to those reserved for other program sections.

In order to verify the correct execution of a program, a key instruction or block of instructions can be monitored. When an instruction fetch occurs from the monitored instructions, the programmer can verify whether the execution up to that point was satisfactory. Similarly, a key data word can be used to verify correct program execution.

Definition of Monitored Area

A core storage area is defined by placing an address in each of two address boundary registers. This area includes the word specified by the address in the lower boundary register and all words in subsequent locations up to, but not including, the address in the upper boundary register. The address in the upper boundary register is normally larger than that in the lower boundary register. If it is equal or smaller, no words are contained in the core storage area. The upper boundary and lower boundary registers are located in bits 0-17 and 32-49, respectively, of storage location 3.

The boundary control bit, bit 57 of storage location 3, determines which condition will cause an address monitoring signal. When the boundary control bit is zero, a signal is obtained for a comparand inside the specified area. When the boundary control bit is one, a signal is obtained for a comparand outside the specified area. An instruction or data field which straddles a boundary always gives an address monitoring signal.

The monitoring of references to locations 0-31 is independent of the contents of the boundary registers. Locations 1, 2, and 3 are referred to as the permanently protected area of storage. References to these locations always give an address monitoring signal, regardless of the contents of the boundary registers or the setting of the boundary control bit. References to locations 0 and 4 through 31, inclusive, never cause an address monitoring signal.

Action of Address Monitoring Signal

The action of the address monitoring system depends upon the state of the interruption system and upon the masks of the indicators that can be activated by the address monitoring signal. The indicators that can be turned on by the presence of the address monitoring signal are DS (data store), DF (data fetch), and IF (instruction fetch).

When the interruption system is disabled, address monitoring is ineffective. In the disabled mode, core storage protection as specified by the address boundary registers in conjunction with the boundary control bit does not take place, and the address monitoring signal is ignored. As a result, the DS, DF and IF indicators cannot be turned on when the system is disabled.

When the interruption system is enabled, any reference to the protected storage area causes one of these three indicators to be turned on, the choice depending upon the type of storage reference. The subsequent action depends upon the status of the mask bit. If the mask bit is one, the operation is terminated and an interruption occurs; if the mask bit is zero, the indicator is set, but the operation proceeds. The subsequent description of the action of the address monitoring system applies only when the interruption system is in the enabled state.

When an address alarm is caused by a data address used in a store-type operation, indicator DS is turned on. Since this indicator is permanently masked on, storing in core storage is always suppressed, and the operation is always terminated. A store-type operation is one in which the contents of an addressed storage location are subject to change.

When an address alarm is caused by a data address used in a fetch-type operation, indicator DF is turned on. If the DF mask bit is one, the data fetch is suppressed and the operation terminated. If the DF mask bit is zero, the operation proceeds normally. A fetch-type operation is one in which the contents of an addressed storage location are read out for use but are not subject to change by the operation.

When an address alarm is caused by the location of the instruction which is to be executed next, indicator IF is turned on. If the indicator is masked for interruption, the interruption occurs before execution of the instruction causing the indication and after execution of all preceding instructions has been completed or terminated. If the branch address of a successful branch operation is such that control would pass into a protected area, the entire branch operation, including any counting, bit changing, or index alteration, is suppressed. The address boundaries are ignored if the mask bit is zero.

The three address alarm indicators, DS, DF, and IF, are permanent. Once turned on, they remain on until they cause an interruption or are turned off by program control.

When an interruption occurs, the resulting interruption address is not monitored for instruction fetch. Consequently, the interruption table of instructions need not be in the same area of storage as the program. After the instruction at the interruption address is obtained, it is monitored in the usual manner for an instruction of its type.

Addresses Monitored

All storage addresses actually used by the CPU in the course of an operation are subject to address monitoring. If an address which is associated with an operation is not actually used, it will not be monitored by the contents of the boundary registers. For example, the store instruction counter address prefixed to a branch operation is not monitored unless the branch is successful.

Core storage addresses used by the exchange are not subject to address monitoring.

An address which actuates indicator AD, address invalid, is not compared with the contents of the boundary registers and cannot actuate any of the addresss alarm indicators.

Address monitoring for Instruction Fetch includes:

1. The location of each instruction to be executed with the exception of the single instructions at the effective interruption address which are executed when an interruption occurs.
2. The branch address of all successful branch operations.
3. The location of the subject instruction of the EXECUTE and EXECUTE INDIRECT AND COUNT operations when this location is in main storage.

Address monitoring for Data Store includes:

1. The effective address of all store-type operations.
2. The address of the location which receives the original contents of the index register specified in the RENAME operation.
3. All addresses encountered during the SWAP operation.
4. Each address that receives data during the TRANSMIT operation.
5. The effective address of EXECUTE INDIRECT AND COUNT.

Address monitoring for Data Fetch includes:

1. The effective address of all fetch-type operations with a direct address.
2. The effective address at each level of the LOAD VALUE EFFECTIVE operation with the exception of the final effective address which is loaded into the specified index register.
3. The refill address used in index refill operations and the operations REFILL and REFILL ON COUNT ZERO.

4. All addresses encountered during the SWAP operation.
5. Each address from which data are obtained in the TRANSMIT operation.

PROGRAMMING NOTES

In general, when a protected address is encountered during an operation with the interruption system enabled and the corresponding indicator masked for interruption, the operation is terminated. This may or may not result in complete suppression of the operation. In a multilevel operation, such as TRANSMIT, part of the operation may already have been completed when the address alarm occurs, and hence this part of the operation cannot be suppressed.

If any address encountered during the SWAP operation is in a protected area of storage, indicators DS and DF are both turned on and the operation is terminated.

Two addresses are used in the RENAME operation: the effective address and the address in the refill field of X0. The former is monitored for DF and the latter for DS. If indicator DF is turned on and its mask bit is one, or if indicator DS is turned on, the entire operation is suppressed. If the operation is not allowed for either reason, the specified index is neither stored nor loaded.

Since the execute-type operations are used to perform other operations, it is desirable to have the indicators describe the subject instruction as far as possible. For this reason, if the location of the subject instruction is in a protected area of storage, indicator IF is turned on.

If the LOAD VALUE EFFECTIVE operation is terminated at any level because of DF, the specified index is not loaded and remains unchanged. The index flag and index result indicators remain as set before the operation.

The bit address used in the BRANCH ON BIT operation is monitored for an address alarm. The operation is considered a data-fetch operation if it specifies that the tested bit remain in its original state, zero or one. It is considered a data-store operation if it specifies that the tested bit be inverted, or made zero or one regardless of the original setting of the addressed bit.

CONNECT TO MEMORY is a store-type operation regardless of the connective specified.

Not included in the address monitoring are:

1. All data references to storage locations 0 and 4 through 31.
2. Immediate addresses.
3. Effective addresses of LOAD VALUE WITH SUM operations.
4. Channel and control word addresses in input-output operations.
5. Data and control word addresses generated by the exchange.
6. Effective interruption addresses.
7. Effective addresses of EXECUTE operations when these addresses are less than 32.

Note that while index refill operations in the computer are monitored, control word refills which take place in the exchange are not.

Storage Assignment

The address field of all operations provides for an 18-bit word address. Addresses 0 through 31 are for specific purposes assigned to these locations.

Instructions cannot be executed from storage locations 0-31. Any attempt to obtain an instruction from these locations will turn on indicator AD, address invalid. Similarly, any attempt to branch to these locations will turn on indicator AD, and the branch and all associated operations will be suppressed.

PROGRAMMING NOTE

Since instructions cannot be obtained from locations 0 to 31, the address in the interruption address register must be 32 or greater. If it is not, the effective interruption address formed when an interruption occurs may be less than 32. In this event the computer will be unable to obtain an instruction and indicator AD, address invalid, will be turned on. If this, in turn, causes an effective interruption address of less than 32, the computer will not be able to complete the interruption and will hang up at this point. The operator must use the INITIAL PROGRAM LOAD key to override the error condition.

Zero

The 64 bits of storage location 0 are always zero. When information is taken from location 0, an all-zero word is obtained at all times. After an initial power on, this location may contain random bits. Any STORE operation to location 0 causes it to contain a 0 with correct parity. Information stored in location zero can not be recovered. Address 0 can be used as a convenient source of zeros. It also permits pseudo-store operations for test purposes.

Location 0 is a valid word address in all but the following cases, in which it will cause an address-invalid or exchange-program-check indication:

1. Instructions cannot be executed from location 0 either normally or as part of an EXECUTE INDIRECT AND COUNT operation. The contents of this loca-

tion can be executed as the subject instruction of an EXECUTE operation, however.
2. Location 0 is not a valid refill address for control words used in input-output operations.
3. The control word for an input-output operation cannot be taken from location 0.
4. Data cannot be transmitted to or from location 0 in an input-output operation.
5. Execute indirect pseudo-instruction counters may not occupy location 0.
6. The STORE INSTRUCTION COUNTER instruction may not store into location 0.
7. The effective address of the READ, WRITE, and COPY CONTROL WORD instructions may not refer to location 0.
8. Neither the old nor the new name in a RENAME instruction may refer to location 0.

Special Registers

Addresses 1 through 15 are used for all addressable special registers employed by the central processing unit.

Except as noted in connection with the individual registers, addresses 1-15 are valid word addresses in all but the following cases, in which they cause an address-invalid or exchange-program-check indication:

1. Instructions cannot be executed from the special registers, either normally or as part of an EXECUTE INDIRECT AND COUNT operation. The subject instruction of an EXECUTE operation may be located in the special registers, however.

2. The special registers are not valid refill addresses for index or control words.

3. The control word for an input-output operation cannot be located in a special register.

4. Data cannot be transmitted to or from the special registers in an input-output operation.

5. The instruction counter cannot be stored in a special register.

6. The pseudo-instruction counter used in EXECUTE INDIRECT AND COUNT cannot be located in a special register.

7. In the operation RENAME, neither the old nor new name of the index register can be in the range 1 through 15.

Not all bit positions in locations 0-15 are assigned to specific functions. Those bit positions that are not so assigned are always zero. Information stored in these positions is not recoverable. (However, for an exception see the programming note in the next section.)

Interval Timer

The interval timer is intended to measure elapsed time over relatively short intervals. It can be set to any value at any time, and indicator TS, time signal, is actuated when the time period has ended.

The value of the interval timer reading is stored in core storage and occupies bit positions 0 through 18 of location 1. It is continually stepped down by pulses originating from a stable oscillator. Each time the oscillator delivers a pulse, the value of the timer is read out, decremented by one, and returned to core storage. The oscillator operates at 1,024 (2^{10}) cycles per second, or a pulse about every millisecond. Bits 0-8 show time in seconds. A full cycle is about 8½ minutes.

The timer runs whenever the machine is operating, which includes also the execution of BRANCH ENABLED AND WAIT. Whenever it goes from one to zero, it turns on the time signal indicator in the indicator register. The next oscillator pulse then sets the timer to all ones unless the program in the meantime has changed its contents. The timer does not stop after reaching zero.

The interval timer is in the permanently protected area of storage, and cannot be read if indicator DF, data fetch, is masked on and the interruption system is enabled, nor can it be set if the system is enabled. The only store-type operations that may store in the left half of location 1, and hence may be used to set the interval timer, are STORE VALUE, STORE COUNT, STORE REFILL, and STORE VALUE IN ADDRESS, and these operations can be performed only when the interruption system is disabled. When enabled, these operations will turn on indicator DS, data store, and be suppressed. An attempt by any other operation to store into the first half of location 1 will be suppressed and indicator AD, address invalid, will be turned on.

PROGRAMMING NOTE

Core storage provides valid and usable storage locations for each bit position of location 1, including bit positions 19 through 27 which are unused and appear between the interval timer and the time clock. When the interval timer is reset, storing into these bit positions is not suppressed, and, consequently, it is possible that some information may appear there temporarily. Specifically, when STORE VALUE is used to reset the interval timer and bits 19 through 24 of the value placed in the interval timer contain any one-bits, these bits are set into the corresponding bit positions of location 1, and remain there until the first time the interval timer is stepped down. The length of the address modified by STORE VALUE IN ADDRESS depends upon the contents of bit position 24 of the word being changed. Once bit position 24 of location 1 has been set to one, bit positions 19 through 23 can be changed also by means of STORE VALUE IN ADDRESS. A store address of 1 in the instructions T, SWAP, SX, Z, R, and RCZ, causes both the AD and DS indicators to be set if the interrupt system is enabled.

Time Clock

The time clock is provided to measure time difference or duration over relatively long periods. This clock consists of a number 36 bits long which is continually stepped up by pulses originating from the same 1024 cycle per second oscillator which controls the updating of the interval timer. The two clocks are updated consecutively in the same time interval. The leftmost 26 bits of the time clock measure time in seconds. A full cycle is about 777 days.

The value of the clock occupies bits 28-63 of location 1. Each time the oscillator delivers a pulse it is read out, incremented by one, using the index adder, and returned to core storage.

The clock runs continually while the computer is under program control, including the time the computer is executing a BRANCH ENABLED AND WAIT operation. When the clock reaches its maximum reading of all ones, the next oscillator pulse sets it to all zeros. No indication is given when the clock recycles to zero.

The time clock is also in the permanently protected area of storage. It may be read under the same conditions as the interval timer. Its setting, however, cannot be altered under program control. Any attempt to store into the right half of location 1 is suppressed, and indicator AD, address invalid, is turned on.

Programming Notes

The time clock can be used to obtain a time-of-day indication. A known external time is taken as a reference point and the setting of the time clock at that time is stored. The time-of-day at a later time can be obtained by using the time which has elapsed since the reference time. The difference between the current clock setting and the setting at the time of reference can be converted to hours, minutes and seconds. If this time difference is added to the time-of-day which was used as the reference, the current external time of day is obtained. Since the time clock is continually stepping and has a recycle time of over two years, it may be used to provide a convenient "serial number" for program outputs. Each output can include a clock reading which will provide a chronological identification of the output.

Interruption Address

Bits 0-17 of location 2 form the interruption address register. When an interruption occurs, the contents of this register are used to obtain the address of an instruction to be inserted into the normal sequence. The interruption address register is located in the permanently protected area of storage.

Address Boundaries

The address monitoring system requires the definition of an area in core storage. The two limits of this area are defined by the upper and lower boundary registers. The area starts with the word at the address in the lower boundary register and includes all words in subsequent locations up to, but not including, the word at the address in the upper boundary register. If the contents of the upper boundary register are equal to or smaller than the contents of the lower boundary register, no words are included in the area.

The upper boundary register occupies bit positions 0-17 of location 3. The lower boundary register occupies bit positions 32-49 of location 3. The boundary control bit, which determines whether addresses inside or outside of the defined area are to be protected, is located in bit 57 of location 3. All bits of location 3 are in the permanently protected area of storage.

Maintenance Bits

The 64 bits of storage location 4 are reserved for maintenance purposes. These bits are active only when the machine is in the maintenance mode. When it is not in this mode, they act like the bits of location 0. After an initial program load, this location contains the first word read from the input unit. This is the control word for the initial load. Any store to location 4 causes it to contain a 0 with correct check bits.

Channel Address

The exchange channel responsible for the current settings of the input-output status indicators is identified by the channel address register. This register occupies bits 12-18 of location 5. The channel address register is a read-only register; any information stored there is lost. In this respect it resembles location 0. The channel address register can be set only by the exchange.

Other CPU Bits

In some systems, several computers or other central processing units may be very closely interrelated, sharing common core storage units or a common exchange. In order to permit a program in one computer to signal others, the other CPU register, bits 0-18 of location 6, and the CPU signal indicator, CPUS, have been provided. Each of the bits can actuate the CPU signal indicator of one and only one other computer. The system thus provides for up to nineteen other computers. When a bit is set to one by programming, the CPU signal indicator is actuated in the computer associated with that bit. In this manner any computer can actuate the indicator in any other computer, and thus interrupt the other computer's program. When a bit is zero, it has no effect on the associated computer.

The other CPU register is provided only in computers which are to be used in multi-computer systems. When not present, these bit positions behave like the bit positions of address 0.

PROGRAMMING NOTE

Consider a system with ten closely coupled computers known as CPU 0, CPU 1, ... CPU 9. If the program in CPU 4 turns on bit 6.07 in its registers, the CPU indicator in CPU 7 is actuated, permitting program interruption in machine 7. The program which responds to such interruption examines some agreed-upon location in the common core storage to find the full message that CPU 4 desired to send. This message may contain the fact that it is sent by CPU 4.

This technique is necessary only in closely coupled multi-computer systems. In systems where each computer has its own core storage units and exchange, the several computers can communicate with one another through the exchanges, using channel signal to attract the attention of the called computer.

Left Zeros Counter

In the connective operations, a count is developed of the number of zeros in the result field which are to the left of the most significant one-bit. This count is placed in the left zeros counter, bits 17-23 of location 7. It remains there until changed by another connective operation or by one of the operations described below.

The two high-order bits of the left zeros counter are also used to distinguish between the four operations that can turn on the indicator DT, decimal transit. These are the decimal operations MULTIPLY, DIVIDE, MULTIPLY AND ADD, and LOAD TRANSIT AND SET. The remaining bits of the left zeros counter are cleared. The left zeros counter is similarly set when indicator BT, binary transit, is turned on by a binary LOAD TRANSIT AND SET operation.

The contents of the left zeros counter are also changed in all floating point division operations. In these operations the left zeros counter contains the amount of shift that is applied in the normalization process.

All-Ones Counter

In the connective operations, a count is developed of the total number of ones in the result field. This count is placed in the all-ones counter, bits 44-50 of location 7. It remains there until changed by another connective operation or by one of the operations described below.

The operations that turn on indicator BT, binary transit, or indicator DT, decimal transit, also place the effective offset of the operation in the all-ones counter. These operations are LOAD TRANSIT AND SET and decimal MULTIPLY, DIVIDE, and MULTIPLY AND ADD.

Accumulator and Sign Byte Register

In many operations an implied operand is a field from the accumulator. The accumulator is a 128-bit register occupying locations 8 and 9. The left half of the accumulator is in location 8, and the right half is in location 9. The sign of the accumulator operand is contained in an associated register, the accumulator sign byte register, an 8-bit register occupying bits 0-7 of location 10.

Variable field length binary quantities may be positioned at any point in the accumulator. In decimal operations, the accumulator is considered as 32 digit positions, each of four bits. A variable field length decimal quantity may be positioned at any of these digit positions, but its digits may not be split between accumulator digit positions.

Floating point quantities in the accumulator may have either single- or double-precision format. In either format the exponent and its sign occupy bits 0-11 of the left half of the accumulator, location 8. In the single-precision format, the 48-bit fraction occupies bits 12-59 of the left half of the accumulator. The remainder of the accumulator is not used. In the double-precision format, the high-order portion of the fraction occupies the same position as the single-precision fraction. The remaining low-order 48 bits of the 96-bit double precision fraction extend from bit 60 of the left half of the accumulator, location 8, through bit 107 of the right half of the accumulator, location 9. The sign of the fraction is located in the accumulator sign byte register in either format.

Operands in storage may have associated with them sign bytes ranging in size from one to eight bits. The eight possible sign bytes and their corresponding positions in the accumulator sign byte register are:

BYTE SIZE	SIGN BYTE	POSITION IN SIGN BYTE REGISTER							
		0	1	2	3	4	5	6	7
1	S	0	0	0	0	S	0	0	0
2	S T	0	0	0	0	S	T	0	0
3	S T U	0	0	0	0	S	T	U	0
4	S T U V	0	0	0	0	S	T	U	V
5	Z S T U V	0	0	0	Z	S	T	U	V
6	Z Z S T U V	0	0	Z	Z	S	T	U	V
7	Z Z Z S T U V	0	Z	Z	Z	S	T	U	V
8	Z Z Z Z S T U V	Z	Z	Z	Z	S	T	U	V

Here S is the sign bit, T, U, and V are flag bits, and the bits marked Z are zone bits. The sign and flag bits may be placed in or obtained from the 8-bit accumulator sign byte register. The sign byte is positioned in such a way that the sign bit always occupies position 4 in the register. The zone bit positions, 0-3, of the sign byte register are not changed in an arithmetic operation unless specifically addressed. The zone bits of a sign byte do not normally enter the sign byte register. In data store operations, the zone bits of the sign byte register may be used.

Normally, the accumulator contents are addressed as the implied operand of an operation. In this case, the sign, flag, and zone bits of the accumulator sign byte register are correctly associated with this operand.

The accumulator may also be addressed as an operand in storage. When location 8, the left half of the accumulator, is explicitly addressed as the operand in a floating point operation, a word is obtained that consists of bits 0-59 of the accumulator followed by bits 4-7 of the accumulator sign byte, thus forming a proper floating point word.

When locations 9 or 10 are explicitly addressed as the operand of a floating point operation, or any part of the accumulator or the accumulator sign byte is explicitly addressed as the operand of any other operation, these locations are treated like any other storage locations. Thus, in a variable field length operation, the contents of the accumulator sign byte register are interpreted correctly as the zone, sign, and flag bits of the accumulator contents only if byte size 8 is specified in the operation, and the low-order bit of the addressed field is bit 7 of location 10.

Indicator Register

The indicator register, location 11, contains 64 indicators. Each indicator is turned on (set to 1) or off (set to 0) when certain conditions unique to it occur in the computer system. When an indicator is on, the corresponding bit in the mask register is 1, and the interruption system is enabled, an automatic interruption will occur. When an indicator causes an interruption, it is automatically turned off. An indicator may also be turned off by a BRANCH ON INDICATOR instruction that tests it.

Bit positions 0-19 of the indicator register can be read, but not loaded. Any attempt to store into these positions has no effect. Bit positions 20-63 may either be read or loaded.

When the indicator register is addressed as the operand of a fetch-type instruction, its contents at the start of execution of that instruction are obtained. (If indicator IF, instruction fetch, is to be turned on because of the location of an instruction, it will have been turned on when the instruction was fetched, previous to the start of execution of that instruction.) When an instruction calls for storing into any part of the indicator register, the result in bits 20-63 of the indicator register will be the same as if this operation had acted upon any other storage location whose initial contents were the same as those of the indicator register at the start of execution of the instruction. The setting of any of the indicators 20-63 that would normally occur as part of the store operation is overridden by this result. In the execution of the operations BRANCH ON INDICATOR and BRANCH ON BIT when the interruption system is enabled, the actuation of indicator IF that results if a branch is attempted to a protected location is considered to be associated with the instruction at that location, and hence in such cases this indicator will be on at the completion of the operation regardless of the previous setting of this indicator or any alteration of it specified in the operation.

Mask Register

Corresponding to each indicator in the indicator register is a bit position in the mask register, location 12. Those indicators for which the corresponding mask bit is zero cannot cause automatic program interruption. Those indicators for which the mask bit is one may cause interruption if the interruption system is enabled.

Bits 0-19 of the mask register are permanently set to one. Bits 20-47 can be set to either zero or one by storing the desired value in these positions. Bits 48-63 are permanently set to zero. Any information stored in bit positions 0-19 or 48-63 of location 12 is lost and is not recoverable.

Remainder Register

The remainder developed in the binary integer DIVIDE operation or the floating point DIVIDE DOUBLE operation is placed in the 64-bit remainder register, location 13. In the binary integer operation, the four low-order bits of the register form a 4-bit sign byte in which the flag bits are zero. In the floating point operation, the format of the remainder is the same as that of a floating point word in storage whose flag bits are zero. The remainder register is not altered as the implied operand of any other operations.

Factor Register

The operation MULTIPLY AND ADD obtains one of the factors in the multiplication from the factor register as an implied operand. The factor register is a 64-bit register in location 14.

The factor register is loaded as an implied operand in the operation LOAD FACTOR. In an integer operation, the four low-order bits of the factor register form a 4-bit sign byte in which the flag bits are zero. In a floating point operation, the factor has the same format as a floating point word in storage whose flag bits are zero. The factor register is not altered as the implied operand of any other operation.

Transit Register

The 64-bit transit register, location 15, is used to provide a linkage to the subroutines that handle interruptions caused by the transit indicators. This register is loaded as part of the operation LOAD TRANSIT AND SET and also the decimal operations MULTIPLY, DIVIDE, and MULTIPLY AND ADD, which are very similar in action to LOAD TRANSIT AND SET.

The operation LOAD TRANSIT CONVERTED also loads the transit register, placing the result of the radix conversion in that location.

In each of the operations above, the four low-order bits of the transit register form a 4-bit sign byte. The transit register is not altered as the implied operand of any other operations.

Index Registers

The sixteen index registers X0 through X15 have memory addresses 16 through 31. Index register X0 cannot be used for address modification. It can, however, be used in progressive indexing and as an operand location in index arithmetic. The other 15 index registers can be used in any index operation. The refill field of index register X0 is also used in the operation RENAME to hold the name of another index register.

Addresses 16-31 are valid word addresses in all but the following cases, in which they cause an address-invalid or exchange-program-check indication:

1. Instructions cannot be executed from the index registers, either normally or as part of an EXECUTE INDIRECT AND COUNT operation. The subject instruction of an EXECUTE operation may be located in an index register, however.
2. The control words for an input-output operation cannot be located in an index register.
3. Data cannot be transmitted to or from the index registers in an input-output operation.
4. In the operation RENAME, neither the old nor new name of the index register can be that of an index register.

Main Core Storage

Addresses from 32 through 262,143 ($2^{18}-1$) are available for addressing the main core storage. Not all of these addresses may be provided in a given installation. If less than the maximum amount of core storage is provided, consecutive addresses starting at location 0 are used, of which the first 32 are reserved for special registers, and hence not used in main core storage. The entire address is used from the instruction, and if the effective address is above the limit of available core storage, indicator AD, address invalid, or indicator EPGK, exchange program check, will be turned on. Locations in main core storage are valid word addresses in any operation.

Normal Sequential Operation

Normally, the operation of the computer is controlled by instructions taken in sequential order. An instruction is fetched from a core storage location whose address is specified by the contents of the instruction counter. Any required address modification on the instruction is then performed and when execution of the previous instruction is sufficiently completed, execution of the modified instruction is begun. Instructions are taken in sequences other than normal as a result of branching operations, interruptions, and execution operations.

Instruction Counter

The instruction counter is a 19-bit register that specifies the location of the next instruction to be executed. When the nineteenth bit is a one, the full word taken from storage will start at bit 32 of the addressed location and continue to bit 31 of the next higher location. When the nineteenth bit is a zero, the full word will start at bit 0 and continue to bit 63 of the addressed location.

Some instruction words are full length occupying 64 bits while others are half length of 32 bits. Both full and half-word instructions may be intermixed in storage. Each time an instruction is executed, the instruction counter is stepped to specify the location of the next instruction. If the fetched instruction is half length, the instruction counter is stepped once at its nineteenth position. If full length, the counter is stepped twice.

The instruction counter is not addressable by the program, but its contents can be changed by a branch-type operation or stored at any storage location by prefixing a half-length branch instruction with the half-length instruction STORE INSTRUCTION COUNTER IF.

PROGRAMMING NOTE

If an instruction causes its own modification (other than that caused by index registers), that modification changes only the instruction format in storage and does not alter the execution in process. If the instruction modifies any succeeding instruction, the modification is effected before the execution of that instruction.

Branching

The normal sequence of instructions may be altered by the use of branching instructions. These are of four types: unconditional, indicator, index, and bit branching. Any of these instruction types may cause a new value to be transferred to the instruction counter, thereby causing the instruction sequence to proceed from the newly addressed location.

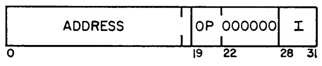
The addresses between 0.0 and 31.32 are invalid branching addresses. All other available 19-bit addresses are valid.

PROGRAMMING NOTES

After an instruction is fetched, but before its execution, the instruction counter contents are stepped in ordinary fashion. Accordingly, with branching operations, this is done prior to any attempt to replace the instruction counter contents. For example, if the instruction counter contents are 231.32 and a half-word branch instruction is fetched from the addressed location, before execution of the fetched instruction, the instruction counter is stepped to 232.0.

Unconditional Branching

There are five unconditional branching operations, all of which provide for normal modification of the branch address by the contents of an index register. Unconditional branching instructions are specified in the half-word format shown. The address field, bits 0-18, represents the branch address. Unconditional branching is specified by bits 19-27, where bits 19-21, the OP field, select the particular operation. Bits 28-31 name the index register used for address modification.



Branch (B)

This operation replaces the instruction counter contents with bits 0-18 of the effective address of the instruction.

Branch Relative (BR)

This instruction causes the absolute value of the effective address to be added to the instruction counter contents. Bits 0-18 of the sum replace the instruction counter contents.

Branch Enabled (BE)

This operation enables the interruption mechanism, and then performs the same operation as BRANCH.

PROGRAMMING NOTE

A half-word of all zeros when taken as an instruction will be interpreted as BRANCH ENABLED to location 0.0, an invalid address.

Branch Disabled (BD)

This operation disables the interruption mechanism, and then performs the same operation as BRANCH.

Branch Enabled and Wait (BEW)

This instruction provides a means of stopping program execution. The interruption mechanism is enabled, and the instruction counter contents are replaced with bits 0-18 of the effective address of the operation as in BRANCH ENABLED. However, no further instruction will be executed until after an interruption occurs.

No Operation (NOP)

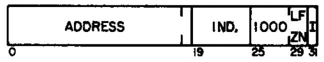
The address and index fields of NO OPERATION are not interpreted, so they cannot cause any indicators to be actuated; no addressable bits or locations are altered. The computer proceeds to the next instruction in sequence.

PROGRAMMING NOTE

The operation code for NO OPERATION differs from those for BRANCH and BRANCH ENABLED AND WAIT in only one bit position each. Similarly, the operation code for STORE INSTRUCTION COUNTER IF NO OPERATION differs in only one bit position from BRANCH ON BIT. Thus, a branching or waiting instruction can readily be switched from active to inactive condition within a program.

Indicator Branching

In indicator branching, the success of branching depends upon a test of one of the 64 indicators. Normal modification of the branch address is provided by index register X1 (address 17). The indicator branching instruction is specified in the half-word format shown below. Bits 0-18, the address, represent the branch address. Bits 19-24 specify the number of the tested indicator. Bits 25-30 specify the operation, where bits 29 and 30 serve as modifiers. Bit 31 determines whether or not there will be branch address modification using index register X1.

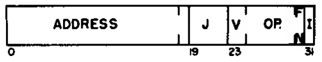


Branch on Indicator (BI)

This operation causes the specified indicator to be tested. If its value matches that of bit 30 of the instruction, bits 0-18 of the effective address replace the instruction counter contents. Whether or not a branching operation is successful, the tested indicator is reset to zero when instruction bit 29 is one; when bit 29 is zero, the tested indicator remains unchanged. A BI which addresses indicators 0 through 19 of the indicator register may not be used as free (due to an interrupt) instruction in the interrupt table.

Index Branching

In index branching, the count field of an index register J is counted down, which sets the index count zero indicator, XCZ, according to whether or not the count field became zero. This setting of XCZ determines the success of branching. Limited incrementing of the value field of the index register is possible. Normal branch address modification can be provided by index register X1. The index branching instructions are specified in the half-word format shown below. Bits 0-18, the address, represent the branch address. Bits 19-22 specify the index register J whose count field will be counted down. Bits 23-30 specify the operation, where bits 23, 24, 29, and 30 are modifiers. Bit 31 determines whether or not there will be branch address modification using index register X1. When bit 29 is zero, the instruction is COUNT AND BRANCH; when the bit is one, the instruction is COUNT, BRANCH, AND REFILL.



Count and Branch (CB)

The count field of index register J is counted down by subtracting one at bit position 45, and appropriately setting the index count zero indicator, XCZ. The value field of index register J is incremented as indicated by the table below:

INSTRUCTION BITS 23	24	CODE	DESCRIPTION OF INDEX VALUE INCREMENTING
0	0		No incrementing.
0	1	H Add half to value	Add one in bit position 18.
1	0	+ Add one to value	Add one in bit position 17.
1	1	— Subtract one from value	Subtract one in bit position 17.

The incrementing takes into account the sign, bit 24, of the value field of the index register. Then, if the status of indicator XCZ matches that of instruction bit 30, bits 0-18 of the effective branch address will replace the instruction counter contents. Indicator XCZ cannot be reset as in BRANCH ON INDICATOR.

Count, Branch, and Refill (CBR)

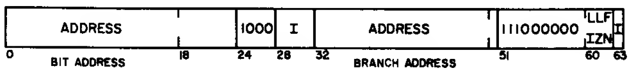
This operation is the same as COUNT AND BRANCH except that, after execution, index register J is refilled if the index count zero indicator is on (count reached zero). The index register is refilled by the contents of the location specified by the refill field of index register J. All available 18-bit addresses larger than and including 16.0 are valid for refilling.

Programming Notes

All index incrementing and comparing operations set indicators that can be tested by the BRANCH ON INDICATOR operation. The generalized functions of incrementing, counting, conditionally refilling, and testing an index register normally require two half-length instructions, such as ADD TO VALUE and BRANCH ON INDICATOR. For the common case, when the increment is + 1, + ½, 0, or — 1, the index branching operation permits all these functions to be specified in a single half-length instruction.

Bit Branching

Any bit in storage can control branching with the BRANCH ON BIT instruction. The choice of the tested bit may be varied by address modification, and the bit may be set to zero, inverted, or set to one if desired. The bit branching instruction is specified in the full-word format shown below. Bits 0-23 constitute the bit address. Bits 24-27 and 51-62 designate the operation, where bits 60-62 are modifiers. Bits 28-31 name the index register to be used for bit address modification. Bits 32-50 represent the branch address. Bit 63 determines whether or not there will be branch address modification using index register X1.



Branch on Bit (BB)

The bit specified by the effective bit address is tested. If its status matches that of bit 62 of the instruction, bits 0-18 of the effective branch address replace the instruction counter contents.

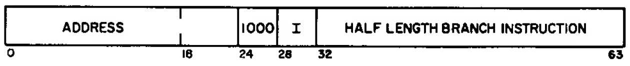
Following the above operation, whether branching occurs or not, the tested bit is set according to the table below:

INSTRUCTION BITS 60	61	TESTED BIT WILL BE
0	0	Unchanged.
0	1	Set to zero.
1	0	Inverted.
1	1	Set to one.

While any bit in storage can be tested by BRANCH ON BIT, those bits in read-only locations will remain unchanged when there is any attempt to change them.

Storage of Instruction Counter

Any half-length branching operation defined above can be converted to a full-word instruction, which also provides for storing the instruction counter at any specified storage location. This is done by prefixing any of the above instructions except BRANCH ON BIT with the operation STORE INSTRUCTION COUNTER IF, as in the format shown below. Bits 0-23 constitute the storage address. Bits 24-27, together with the half-word branch instruction operation code, specify the full-word operation. Bits 28-31 name the index register used for storage address modification. Bits 32-63 specify the half-word branch instruction used.



Store Instruction Counter If (SIC)

This instruction may always be prefixed to one of the half-word branching instructions. It is executed if and only if the associated branching is successful. When it is executed, it causes the instruction counter contents to be stored in bits 0-18 of the half-word specified by bits 0-18 of the effective storage address. Other bits of the half-word are not disturbed. The stored instruction counter contents always address the instruction whose location is next higher in storage than that of the branch instruction to which STORE INSTRUCTION COUNTER IF is prefixed.

The instruction counter may be stored in any available location with 19-bit address 16.0 or above.

PROGRAMMING NOTES

A simple storage of the instruction counter without branching action is accomplished by STORE INSTRUCTION COUNTER IF BRANCH RELATIVE with a branch address of 0.

The combination of the SIC prefix and NO OPERATION is executed as a valid full-word NO OPERATION; the counter is not stored.

STORE INSTRUCTION COUNTER IF cannot be used by itself as a half-length instruction, and it cannot be used to prefix any instruction other than the half-length branching instructions. If STORE INSTRUCTION COUNTER IF should be accidentally used in either of these ways, the full-word beginning with STORE INSTRUCTION COUNTER IF might be interpreted by the computer as a valid connect, integer, or other type of instruction.

Program Interruption System

The sequence of instructions executed may be altered not only by programmed branch-type operations, but also by the computer itself upon occurrence of certain conditions. These conditions are recorded in a collection of indicators, which can be tested by the BRANCH ON INDICATOR operation. An actuated indicator can cause program interruption without an explicit programmed test.

For many indicators there are mask bits which the programmer can set. If a mask bit is set to zero, occurrence of the corresponding indication will not cause program interruption. If, however, the mask bit is set to one, occurrence of the corresponding indication can cause the instruction sequence to be immediately altered so that a single instruction located at a place unique to the causing condition is executed. Since this instruction may be of the branch type, provision is made for complete sequence rearrangements when unexpected or irregularly timed conditions occur. Some indicators have mask bits that are permanently set to zero or one. Those with permanent mask bits of zero can never cause interruption, but they can be tested by branching operations. Indicators with mask bits permanently set to one will cause interruption whenever both the indicator is actuated and the interruption system is enabled.

In order that interruption correction routines have adequate time to correct conditions causing interruption, or at least to store the machine state at the time of interruption, it is necessary to be able to prevent immediately succeeding interruptions. This is accomplished by controlling an *enabling mechanism.* Upon execution of the instruction BRANCH DISABLED, the mechanism will disable the computer's ability to interrupt a program. The system will remain in the disabled state until execution of one of the instructions BRANCH ENABLED or BRANCH ENABLED AND WAIT. The system is then said to be *enabled,* and interruption will occur whenever both an indicator and its corresponding mask bit are on.

Address monitoring is also controlled by the enabling mechanism. When the system is enabled, monitoring is in force; when disabled, monitoring is suspended. Most indicators may come on at any time whether the system is enabled or disabled.

PROGRAMMING NOTE

The program interruption system is so designed that a program written without detailed knowledge of the feature may be interrupted at any time with control transferred to a supervisory program. The procedure for taking care of the various conditions causing interruption can be a task for the supervisory program. On completing the special procedure, the supervisory program can return control to the interrupted program at the point of interruption.

Indicator Register

The indicator register contains 64 bits. Each bit is turned on (set to 1) or off (set to 0) when certain conditions occur in the computer system. Each bit is also turned off automatically as it causes a program interruption. An individual bit may be turned off by a BRANCH ON INDICATOR instruction that tests it.

The indicator register has address 11. Bit positions 0-19 may be read, but not loaded. Any attempt to store into these bit positions has no effect. Bit positions 20-63 may either be read or loaded. Thus, the entire contents of the indicator register may be stored in any storage location by a transmission-type instruction. Similarly, the register may be loaded from any storage location by a transmission-type instruction, but only bit positions 20-63 will be affected.

On any arithmetic or connective operations in which indicator positions 20-63 are the replaced operand, these indicators are read before they can be set by the result of the arithmetic or connective instruction, with the exception of IF, which can be set before the indicators are read when the location of this instruction has a protected address. The result is placed in the indicators at the end of the operation and is exactly the same as if an operand in core storage were used. This result overrides the normal setting of all indicator positions from 20 through 63 which occurs during this type of operation. BRANCH ON BIT may also be used to test and change the contents of positions 20-63 of the indicator register.

For the purpose of program interruption, the bits in the indicator register have a built-in priority, which decreases from left to right. This only affects interruption when two conditions are present when interruption occurs.

Indicator List

On the list in Figure 11, some indicators are *temporary* and other indicators are *permanent.* The permanent indicators remain on when once turned on, unless they cause program interruption or are otherwise turned off. A temporary indicator only remains on (off) until the performance of the next result which could turn it off (on). It is then changed to correspond to that latest result. The comparison result indicators, for example, are temporary. At any time all three of those indicators are set to show

No.	Mnemonic	Mask	Class	Name
Equipment Check				
0	MK	1	P,H	Machine Check
1	IK	1	P,H	Instruction Check
2	IJ	1	P,S	Instruction Reject
3	EK	1	P,C	Exchange Control Check
Attention Request				
4	TS	1	P,C	Time Signal
5	CPUS	1	P,C	CPU Signal
Input-Output Rejects				
6	EKJ	1	P,S	Exchange Check Reject
7	UNRJ	1	P,S	Unit Not Ready Reject
8	CBJ	1	P,S	Channel Busy Reject
Input-Output Status				
9	EPGK	1	P,C	Exchange Program Check
10	UK	1	P,C	Unit Check
11	EE	1	P,C	End Exception
12	EOP	1	P,C	End of Operation
13	CS	1	P,C	Channel Signal
14		1		Reserved
Instruction Exception				
15	OP	1	P,S	Operation Code Invalid
16	AD	1	P,S	Address Invalid
17	USA	1	P,S	Unended Sequence of Addresses
18	EXE	1	P,S	Execute Exception
19	DS	1	P,S	Data Store
20	DF	m	P,S*,C	Data Fetch
21	IF	m	P,S*,C	Instruction Fetch
*Class S when mask is one; otherwise Class C.				
Result Exception				
22	LC	m	P,C	Lost Carry
23	PF	m	P,C	Partial Field
24	ZD	m	P,C	Zero Divisor
Result Exception - Floating Point				
25	IR	m	P,C	Imaginary Root
26	LS	m	P,C	Lost Significance
27	PSH	m	P,C	Preparatory Shift Greater than 48
28	XPFP	m	P,C	Exponent Flag Positive: Exp. $\geq 2^{10}$
29	XPO	m	P,C	Exponent Overflow: Exp. $\geq 2^{10}$
30	XPH	m	P,C	Exponent High: $2^{10} >$ Exp. $\geq 2^9$
31	XPL	m	P,C	Exponent Low: $2^9 >$ Exp. $\geq 2^6$
32	XPU	m	P,C	Exponent Underflow: Exp. $\leq -2^{10}$
33	ZM	m	T,C	Zero Multiply
34	RU	m	P,C	Remainder Underflow
Flagging				
35	TF	m	T,C	Data Flag T — Reset by variable field length and floating point operations requiring data fetch.
36	UF	m	T,C	Data Flag U
37	VF	m	T,C	Data Flag V
38	XF	m	T,S*,C	Index Flag - Reset by index arithmetic operations
*Class S only during index branching when the mask is one and the system enabled; otherwise Class C.				
Transit Operations				
39	BTR	m	P,C	Binary Transit
40	DTR	m	P,C	Decimal Transit
Program				
41-47	PG0-PG6	m	P,C	Program Indicators Zero through Six
48	XCZ	0	T,C	Index Count Zero — Reset by index arithmetic ops. other than comparisons
49	XVLZ	0	T,C	Index Value Less than Zero
50	XVZ	0	T,C	Index Value Zero
51	XVGZ	0	T,C	Index Value Greater than Zero
52	XL	0	T,C	Index Low — Reset by index comparison operations.
53	XE	0	T,C	Index Equal
54	XH	0	T,C	Index High
Arithmetic Result				
55	MOP	0	T,C	To-Memory Op. — Reset by variable field length or floating point op.
56	RLZ	0	T,C	Result Less than Zero — Reset by variable field length or floating point ops. except comparisons
57	RZ	0	T,C	Result Zero
58	RGZ	0	T,C	Result Greater than Zero
59	RN	0	T,C	Result Negative
60	AL	0	T,C	Accum. Low — Reset by comparison operations
61	AE	0	T,C	Accum. Equal
62	AH	0	T,C	Accum. High
Mode				
63	NM	0	P,C	Noisy Mode

T	Indicator temporary and is reset by later operations.
P	Indicator permenent but may be turned off by interruption or during BI.
C	The execution of the instruction during which the indicator is actuated is completed.
H	The execution of the instruction during which the indicator is actuated is terminated.
S	The execution of the instruction during which the indicator is actuated is suppressed, except during EX,EXIC,T, and SWAP.

Figure 11. Indicator List

the result of the most recent comparison performed, unless they have been changed by programming.

The input-output status indicators EPGK, UK, EE, EOP, and CS are controlled as a group, so they collectively describe the status of a single unit at any time. These indicators are set to describe a unit status and the channel address register is set to the address of the unit. As interruptions occur, the indicators are reset. When all five are zero and if the interruption mechanism is enabled, the exchange may set the whole group to describe another unit. This group of indicators is described in detail in the Exchange section.

Definitions of Indicators

The 64 indicators are defined below in terms of the conditions which would actuate the indicator, and the peculiar conditions by which the indicator is turned off. Any indicator is turned off by interruption upon it, and can be turned off or left unchanged when testing it with a BRANCH ON INDICATOR operation. Indicators 20 through 63 may also be turned on or off by a BRANCH ON BIT operation or by operating upon the indicator register contents as a field in storage. Except where noted, these are the only means of turning indicators off. The indicators are listed in the order of their priority.

Equipment Check

0. Machine Check (MK). An error has been detected by the machine checking circuits. There is no guarantee that any specific element in the machine is in either its original or correct state.

1. Instruction Check (IK). An error has been detected during the performance of the current instruction. Only elements usually affected by the operation can be in error. The instruction counter (which cannot be in error) contains the address of the next instruction in storage after the one during which the error was detected.

2. Instruction Reject (IJ). An error has been detected which was identified with the current instruction. The instruction has been executed as though it were a NO OPERATION instruction. The instruction counter contains the address of the next instruction in storage.

3. Exchange Control Check (EK). The exchange has failed to function properly in a manner that is not identified with any particular unit.

Attention Request

4. Time Signal (TS). The interval timer has become zero.

5. CPU Signal (CPUS). Some other instruction execution system in another central processing unit desires the attention of this CPU.

Input-Output Reject

6. Exchange Check Reject (EKJ). An error was detected by the exchange in the course of testing and setting up the present instruction. This reject may indicate equipment malfunction or attempted selection of a channel which is not available to the programmer. The instruction is not executed, and the indicator is actuated before the computer proceeds to the next instruction.

7. Unit Not Ready Reject (UNRJ). An instruction was given for a unit which was not ready to be operated. The unit ready status bit is zero. The instruction is not executed, and the indicator is actuated before the computer proceeds to the next instruction.

8. Channel Busy Reject (CBJ). An instruction was given for a unit which was still connected to the exchange or which is still waiting to give a program interruption as a result of a previous instruction. The instruction is not executed, and the indicator is actuated before the computer proceeds to the next instruction. Certain operations, such as rewinding tape, are completed by the unit after disconnecting from the exchange. New instructions for channels performing such operations can be accepted; they do not cause busy rejects.

Input-Output Status

9. Exchange Program Check (EPGK). The last operation initiated for the unit specified by the channel address register has been terminated by a programming error. Such errors include giving a unit an instruction it cannot execute or specifying core storage locations to which the exchange does not have access.

10. Unit Check (UK). The last operation initiated for the unit specified by the channel address register encountered malfunctioning of the equipment or defects of the recording medium. These malfunctions include uncorrectable data errors, card jams, broken tapes, and so on.

11. End Exception (EE). The last operation initiated for the unit specified by the channel address register encountered an exceptional condition, usually associated with the recording medium or some subdivision of the data to be transmitted, which is not expected to occur during every operation. Examples of such conditions are: out of material, sensing of a tape mark, or depression of the erase key on a console.

12. End of Operation (EOP). The last operation initiated for the unit specified by the channel address register has been completed as specified by the instruction and its control words, if any, unless the unit check indicator is also on. End of operation in conjunction with unit check indicates that the operation was terminated because an uncorrectable data error had been discovered.

13. Channel Signal (CS). A unit on the channel specified by the channel address register has transmitted a channel signal to the computer. If the signal originates while one of the units connected to this channel is in operation, it is transmitted to the computer at the end of the operation. The signal can originate upon depression of the signal key on the unit, by the readying of a unit, or by completion of operations such as rewinding.

14. Reserved

Both the input-output rejects and status indicators are described in greater detail in the Exchange section.

Instruction Exceptions

15. Operation Code Invalid (OP). The instruction just attempted used an operation code or combination of modifiers which is not defined. An instruction with invalid operation code is executed as NO OPERATION.

16. Address Invalid (AD). The location of the instruction next due for execution is at an address not provided in the computer system, or the location specified by the effective operand address of the instruction just attempted is either not provided in the computer system or invalid for the specified operation. The affected instruction is executed as a NO OPERATION. Branching to an address below 32 also actuates the indicator. A negative effective address is not invalid; the magnitude of such an address is used in the operation.

17. Unended Sequence of Addresses (USA). A LOAD VALUE EFFECTIVE, EXECUTE, or EXECUTE INDIRECT AND COUNT operation has been in progress for longer than one millisecond (several hundred indirect addressing cycles). The operation is terminated wherever it may be and the program proceeds. This indicator is the only means of breaking out of a loop that occurs as a result of a programming error in the use of execution operations or indirect addressing. If interruption occurs, it takes place after execution of the causing instruction has been terminated.

18. Execute Exception (EXE). An instruction was executed under control of the EXECUTE or the EXECUTE INDIRECT AND COUNT operations that attempted to change the contents of the instruction counter. The subject operation was suppressed.

19. Data Store (DS). A STORE or other to-memory operation has attempted to change the contents of some location in the protected area of storage while the interruption system was enabled. The operation was suppressed.

20. Data Fetch (DF). A computer operation has attempted to fetch data from a location in the protected area of storage while the interruption system was enabled. If the corresponding mask bit was one, the fetch did not take place, and the entire operation was suppressed. If the mask bit was zero, the operation was completed normally.

21. Instruction Fetch (IF). An attempt has been made to fetch an instruction from, or branch to, a location in the protected area of storage while the interruption system was enabled. If the corresponding mask bit was one, the fetch or branch did not take place and the entire operation was suppressed. If the mask bit was zero, the instruction was executed normally.

Result Exceptions

22. Lost Carry (LC). Information was lost on the operation just executed as a result of a carry propagating beyond the end of the legitimate sum field, as defined for the operation. This includes unsigned add-type to-memory operations which attempt to reverse the sign of the storage field. Lost carry is actuated in unnormalized floating point operation when a fraction overflow bit occurs in a sum or when a low-order one is lost in a remainder. Lost carry is also actuated when a SHIFT FRACTION operation shifts non-zero bits left beyond the end of the fraction field.

23. Partial Field (PF). The operation just executed failed to use all the operand data provided. This condition may arise in any of the following ways:

a. Significant bits of a result extend beyond the left end of the accumulator.

b. A storage-altering operation uses an operand from the accumulator which has non-zero data to its left.

c. The decimal operand of a single-length decimal-to-binary conversion instruction is over 64 bits long when translated to byte size four.

d. A binary multiplier, multiplicand, divisor, or result of a decimal-to-binary LOAD TRANSIT CONVERTED or CONVERT exceeds 48 bits.

e. A binary dividend or result of a CONVERT DOUBLE exceeds 96 bits.

f. In unnormalized floating-point division, the magnitude of the dividend fraction is larger than or equal to the magnitude of the divisor fraction.

24. Zero Divisor (ZD). The divisor or divisor fraction of the division just attempted consisted only of zero bits. The division was not attempted.

Result Exceptions, Floating Point Only

25. Imaginary Root (IR). A STORE ROOT operation was performed on the magnitude of a negative operand.

26. Lost Significance (LS). In a floating point add-type operation in which at least one operand was non-zero, both the final result placed in the accumulator and the overflow bit were zero. This indicator is not turned on when both operands have zero fractions prior to the pre-addition shifting, when the result is a forced zero produced in add to magnitude type operations, or when the result has a propagated exponent flag.

27. Preparatory Shift Greater than 48 (PSH). A floating point add-type operation, using operands with exponent flags of zero, required a preparatory shift of the operands relative to one another by an amount greater than 48 places.

28. Exponent Flag Positive: Exponent $\geqq 2^{10}$ *(XPFP).* The result of a floating point operation had a positive exponent with an exponent flag of 1 propagated from an operand with an exponent flag of 1.

29. Exponent Overflow: Exponent $\geqq 2^{10}$ *(XPO).* The result of a floating point operation had a positive exponent with an exponent flag of 1, generated from operands with exponent flags of 0.

30. Exponent Range High: $2^{10} >$ *Exponent* $\geqq 2^9$ *(XPH).* The result of a floating point operation had a positive exponent with an exponent flag of 0, in the range 2^9 through $2^{10} - 1$, inclusive.

31. Exponent Range Low: $2^9 >$ *Exponent* $\geqq 2^6$ *(XPL).* The result of a floating point operation had a positive exponent, with an exponent flag of 0, in the range 2^6 through $2^9 - 1$, inclusive.

32. Exponent Underflow: Exponent $\leqq -2^{10}$ *(XPU).* The result of a floating point operation had a negative exponent with an exponent flag of 1, generated from operands with exponent flags of 0.

33. Zero Multiply (ZM). The final result of a normalized floating point operation using multiplication is an order of magnitude zero with exponent not in the XFN range.

34. Remainder Underflow (RU). The remainder developed during a floating point division had a negative exponent with an exponent flag of 1, generated from a dividend with exponent flag of 0.

Flagging

35. Data Flag T (TF). The data flag bit T (immediately to the right of the sign bit) of the operand of the instruction just executed was 1. The indicator is turned off by any arithmetic instruction whose operand from storage is not so flagged.

36. Data Flag U (UF). The second of the possible data flag bits of the operand just used was 1. The indicator is turned off by any arithmetic instruction whose operand from storage is not so flagged.

37. Data Flag V (VF). The third and rightmost of the possible data flag bits of the operand just used was 1. The indicator is turned off by any arithmetic instruction whose operand from storage is not so flagged.

38. Index Flag (XF). The index flag bit (bit 25) of the index word just modified was 1, before any refill operation was performed. The indicator is turned off by any index arithmetic or refill operation whose modificand does not have its flag on.

Transit Operations

39. Binary Transit (BTR). A binary LOAD TRANSIT AND SET operation was executed.

40. Decimal Transit (DTR). A decimal MULTIPLY, DIVIDE, MULTIPLY AND ADD, or LOAD TRANSIT AND SET operation was executed.

Program

41-47. Program Indicators Zero through Six (PG0-PG6). These indicators are set by programming only.

Index Result

48. Index Count Zero (XCZ). The result count field of the index word last modified by an index modification operation, including REFILL, was zero before a refill, if any, but after any other modifications. This indicator is turned off by any index modification operation whose result does not have a zero count field.

49. Index Value Less than Zero (XVLZ). The value field resulting from the index modification operation just executed was non-zero and negative. This indicator is turned off by any index modification operation whose result has a value field which is zero or positive.

50. Index Value Zero (XVZ). The value field resulting from the index modification operation just executed was zero. This indicator is turned off by any index modification operation whose result has a value field which is non-zero.

51. Index Value Greater than Zero (XVGZ). The value field resulting from the index modification operation just executed was non-zero and positive. This indicator is turned off by any index modification operation whose result has a value field which is zero or negative. One and only one of indicators XVLZ, XVZ, XVGZ is on at any time, unless they have been changed by programming.

52. Index Low (XL). The result of the index comparison operation just executed was that the compared field of the index word was lower than that of the comparand specified by the effective address. This indicator is turned off by an index comparison whose result is not low.

53. Index Equal (XE). The result of the index comparison just executed was that the compared field of the index word was equal to that of the comparand. This indicator is turned off by an index comparison whose result is not equal.

54. Index High (XH). The result of the index comparison just executed was that the compared field of the index word was higher than that of the comparand. This indicator is turned off by an index comparison whose result is not high.

One and only one of indicators XL, XE, and XH is on at any time, unless they have been changed by programming.

Arithmetic Result

55. To-Memory Operation (MOP). The result of the arithmetic or connective operation last executed was placed in storage. This indicator is actuated by integer and floating point to-memory operations, including STORE, ADD TO MEMORY and its variants, and CONNECT TO MEMORY. It is turned off by any other arithmetic, connective, or comparison operation.

56. Result Less than Zero (RLZ). The result of the integer or floating-point arithmetic operation just executed was non-zero and negative. This indicator is turned off by any non-comparative arithmetic operation whose result is zero or positive, or by any connective operation.

57. Result Zero (RZ). The result of the non-comparative arithmetic or connective operation just executed was zero. This indicator is turned off by any non-comparative arithmetic or connective operation whose result was non-zero.

58. Result Greater than Zero (RGZ). The result of the non-comparative arithmetic operation just executed was non-zero and positive or the result of the connective operation just executed was non-zero. This indicator is turned off by a non-comparative arithmetic operation whose result is zero or negative, or by a connective operation whose result is zero. One and only one of indicators RLZ, RZ, and RGZ is on at any one time, unless they have been changed by programming.

59. Result Negative (RN). The result of the non-comparative arithmetic operation just executed was negative whether zero or not. This indicator is turned off by a non-comparative arithmetic operation whose result is positive, whether zero or not, or by any connective operation.

60. Accumulator Low (AL). The result of the arithmetic comparison operation just executed was that the accumulator contents were less than the comparand specified by the effective address. The arithmetic comparison operations are COMPARE, COMPARE FIELD, COMPARE MAGNITUDE, COMPARE FOR RANGE, COMPARE FIELD FOR RANGE, COMPARE MAGNITUDE FOR RANGE, COMPARE IF EQUAL, and COMPARE FIELD IF EQUAL. This indicator is turned off by a comparison operation whose result is not low.

61. Accumulator Equal (AE). The result of the arithmetic comparison operation just executed was that the accumulator contents were equal to the comparand or within the compared range. This indicator is turned off by a comparison operation whose result is not equal or within the compared range.

62. Accumulator High (AH). The result of the arithmetic comparison operation just executed was that the accumulator contents were greater than the comparand. This indicator is turned off by a comparison operation whose result is not high. One and only one of indicators AL, AE, and AH is on at any time, unless they have been changed by programming.

Mode

63. Noisy Mode (NM). This indicator can only be turned on by program control, through the use of a connective operation, a loading of the indicator register, or BRANCH ON BIT. When on, all floating point operations are performed in the noisy mode, as defined in "Floating Point Arithmetic."

Mask Register

The mask word consists of 64 bits. Each bit corresponds to an indicator in the indicator register. If the bit of the mask is 1 and the interruption system enabled, occurrence of the corresponding condition will cause program interruption at the end of the operation during which the indicator is turned on. If the bit of the mask is 0, turning on the indicator does not cause program interruption. Thus, interruption requires the mask bit to be 1, the indicator bit to be 1, and the mechanism to be enabled.

Not all of the 64 mask bits can be set by program control. Those corresponding to indicators 0-19 are permanently set to 1. Those corresponding to indicators 20-47 can be set to 0 or 1 by program control. Those corresponding to indicators 48-63 are permanently set to 0. Thus, when the interruption mechanism is enabled, some indicators always cause program interruption, some may or may not, and others can never cause interruption. Figure 11 shows the masking property of each indicator.

The mask has address 12 and can be loaded and stored by transmission instructions. When the mask contents are read, the permanently set bits read out as 1 or 0 according to their type, regardless of what may have been loaded into those positions earlier. Individual mask bits with addresses 12.20 through 12.47 can be set or changed by connective operations, arithmetic operations, or BRANCH ON BIT.

Interruption Address Register

Because of the varied conditions which turn on indicators, varied correction routines are required to meet the needs created by the conditions. There can be up to 48 such routines tailored to the program in execution, and each corresponding to an indicator. The initial instruction for a routine occupies all or half of a location in a table of 48 successive full-word locations. The address of the first location is contained in the interruption address register. The interruption address register contains 18 bits and has address 2.0, which places it in the protected area of storage.

Interruption Action

When a program interruption occurs, a mechanism develops the number of the leftmost actuated indicator whose mask bit is one. After this leftmost indicator has been turned off, the developed number is added to the contents of the interruption address register to form the effective interruption address. This address does not replace the contents of the instruction counter but is used to fetch a single instruction which is executed before a second interruption can occur. The instruction fetch indicator (IF) cannot be turned on when this instruction is fetched even though it might lie in the protected area of core storage.

The instruction counter is stepped neither before nor during the execution of the instruction at the effective interruption address; however, the instruction counter can be replaced as the result of a successful branching operation.

When interruption occurs and an instruction at the effective interruption address is executed (it may be of the branch type), the next instruction executed will be one of the following:

1. The instruction then addressed by the instruction counter in case a second interruption has not occurred
2. The instruction at the new effective interruption address in case a second interruption has occurred

If the interruption system is enabled and an indicator and its corresponding mask bit are both on, interruption will follow as early as possible but never during the execution of an instruction. Interruption is permitted after the execution of an instruction is completed, terminated, or suppressed and before execution of the next instruction.

PROGRAMMING NOTES

The leftmost one identification defines the relative priority of actuated indicators if more than one has its corresponding mask bit turned on when interruption occurs. If the interruption instruction does not disable the system in such a case, then another succeeding interruption will be caused by the new leftmost selected indicator before the execution of an instruction addressed by the instruction counter. This is true even if the first interruption instruction is a branch instruction. The instruction counter is not stepped during execution of either interruption instruction.

The instruction at the effective interruption address may be of any type: half-length or full-length; an arithmetic, connective, branch, or any other type.

A half-length instruction in the right half of the full-word location at the effective interruption address will not be executed except by a branch to that location.

When an instruction itself causes an indicator to be turned on and interruption occurs at the end of the operation in progress, the instruction counter contains the location plus one of that instruction even if the instruction is a branch. Instructions at effective interruption addresses are excepted.

Four levels of generality and complexity can be identified in the correction routines selected by the program interruption mechanism.

The simplest routine is a single instruction which is suitable for some conditions, such as input-output or floating-point range exceptions. In this case, the instruction is stored in the proper place relative to the interruption address, and it is executed whenever the condition occurs. After this instruction is executed, the program proceeds with the first instruction after that on which the interrupting condition arose.

The next simplest type of correction routine consists of a subroutine sufficiently short and simple that further interruptions can be excluded while it is in effect. An unconditional STORE INSTRUCTION COUNTER IF BRANCH DISABLED instruction to the subroutine is put at the proper place in the interruption address table, storing the counter in a suitable address. When the subroutine is complete, return is effected by a BRANCH ENABLED instruction that uses the suitable return address.

In the next most general case, the correction routine itself must be subject to further interruptions under control of the same mask and interruption address. The entry in the interruption address table is then a STORE INSTRUCTION COUNTER IF BRANCH instruction to the correction routine. Return from the correction routine is made with BRANCH.

In the most general case, when interruption controlled by a new mask and/or interruption address must be permitted during the subroutine, it is entered with STORE INSTRUCTION COUNTER IF BRANCH DISABLED. The mask register and/or the interruption address register are loaded and the mechanism is enabled. At the end of the subroutine, the mechanism is disabled, the mask register and/or the interruption address register are restored to their original values, and return to the main program is made with BRANCH ENABLED. This procedure permits any conceivable priority system to be set up for interruptions.

Execution Instructions

Three types of instruction sequencing have already been explained: normal sequencing by the stepping of the instruction counter, in which the program *keeps* control of sequencing; branching, in which the program *gives* control to a second program; and interruption, in which a new program *takes* control. Since it is often desirable to permit one program to execute the instructions of a second program without losing control to the second program, a fourth type of instruction sequencing is provided, which *lends* control. This is furnished by means of the two execution instructions.

One instruction, EXECUTE, directly specifies the location of a second instruction, the *subject* instruction, which is executed. The other instruction, EXECUTE INDIRECT AND COUNT, specifies a location which imitates the instruction counter, a *pseudo-instruction counter*. The pseudo-instruction counter addresses the location of a single subject instruction, which is fetched and executed. After the subject instruction has been fetched, but prior to its execution, the pseudo-instruction counter is stepped.

In either of the execution instructions, the subject instruction may be of any type and is performed according to the normal rules for an operation of its type. However, in order that the main program may have full control at all times, the subject instruction is not allowed to change the state of the interruption system or replace the contents of the instruction counter. The execution exception indicator (EXE) is turned on by any branch operation of a subject instruction in which the condition for a successful branch is met. When indicator EXE is actuated, the branch and all associated operations are not performed. The suppressed actions include the disabling in BRANCH DISABLED, the enabling in BRANCH ENABLED or BRANCH ENABLED AND WAIT, the counting and refilling in COUNT, BRANCH AND REFILL, and so on.

If an execution operation specifies an execution instruction as its subject instruction, the subject instruction of the second-level execution operation is obtained. If this, in turn, is still another execution instruction, the procedure is carried to succeeding levels until an instruction that is not an execution instruction is obtained.

When an execution operation is initiated in the main program, the instruction counter is stepped in normal fashion as in all operations of the main program. However, it is never stepped during the execution of any subject instruction even if the subject instruction is again one of the execution instructions.

In general, when the interruption system is disabled, interruption cannot occur and address monitoring is suspended. In order that control may remain entirely with the main program during an execution operation, two special exceptions are made. First, address monitoring is always in force during (1) EXECUTE and (2) EXECUTE INDIRECT AND COUNT, except for the address of the location of the *first level* pseudo-instruction counter. Second, at the completion, termination, or suppression of an execution operation, interruption is always permitted.

PROGRAMMING NOTE

When interruption occurs following an execution operation with the interruption system disabled, it is caused only by the actuated indicator of highest priority whose mask bit is one. Further interruptions cannot occur until either the system is enabled or another execution operation is completed, terminated, or suppressed. Examination for other actuated indicators can be programmed into the correction routine.

Any actuated indicator whose mask bit is on may cause interruption at the completion, termination, or suppression of an execution operation.

Execute (EX)

Bits 0-18 of the effective address form the address of the location of a subject instruction that is fetched and executed. Subject instructions of EXECUTE may occupy any addressable location. An effective address of EXECUTE that is below 32.00 (including addresses between 1.00 and 3.32) is not monitored by the address monitoring mechanism; that is, indicator IF cannot be actuated by these addresses.

Execute Indirect and Count (EXIC)

Bits 0-18 of the full-word addressed by bits 0-17 of the effective address are used as a pseudo-instruction counter. The contents of the pseudo-instruction counter address the location of a subject instruction that is fetched and executed. After the subject instruction has been fetched but prior to its execution, the pseudo-instruction counter is stepped. If the subject instruction is half-length, the pseudo-instruction counter is stepped once; if full-length, twice.

When EXECUTE INDIRECT AND COUNT is a subject instruction in a multi-leveled execution operation, each pseudo-instruction counter involved is appropriately stepped between the fetching and executing of its corresponding subject instruction.

Pseudo-instruction counters may occupy any of the index registers as well as main core storage locations but cannot occupy locations addressed between 0 and 15. Subject instructions of EXECUTE INDIRECT AND COUNT may occupy any available locations with addresses 32.00 and greater.

PROGRAMMING NOTES

The execution instructions may be used not only for supervisory routines, such as tracing programs, but also for parameter determination in subroutine-calling sequences. For example, execution operations in a subroutine may execute parameter-specifying subject instructions whose locations follow that of the branch to the subroutine. The instruction EXECUTE also executes instructions conveniently from the registers. This facilitates table look-up, diagnostic, and other routines.

System Alerts

During the execution of a program, certain conditions arising because of the program or outside influences should cause the computer to alert the program. Five types of alert causes may be recognized:

1. External signals, as from the interval timer, input-output units, or other central processing units.
2. Data exceptions, such as data flags, zero divisors, or negative operands in square root operations.
3. Result exceptions, such as lost carries, partial fields, or floating point exponents within certain ranges.
4. Instruction exceptions, such as instructions which should not or cannot be completed or should signal when they are completed.
5. Machine malfunctions.

These causes are recorded in the indicator register as they occur, thus altering the program. The interruption system allows programmed action on the alerts. The detailed action of the interruption system should permit (1) independent treatment of alerts, and (2) preservation of the state of execution at the time of alert to facilitate program resumption.

Independent Treatment of Alerts

Each alert can be treated independently by programming because of the following three properties:

1. The interruption system may be disabled as a part of the programmed action taken on an alert.

Thus, other alerts that may have occurred simultaneously can be held for action at a later time when the action on the first alert has been completed.

2. One alert does not affect other alerts. However, an alert may cause an instruction in execution to be suppressed or terminated, thus preventing other alerts from occurring. For example, in SWAP, if a protected address is encountered, it will cause termination of the execution. The transmissions that would have been performed if the execution had not been terminated might have encountered an invalid address. Thus, possible alerts are prevented.

3. Each cause for alert actuates a single indicator, except for the input-output status indicators that collectively form a single alert. Furthermore, each alert has a single cause. It is possible, however, for several alerts to occur during the execution of a single instruction. More than one alert can be caused by (1) the instruction in progress, and (2) external signals. For example, the single floating point instruction STORE ROOT can cause the alerts IR and XPH. Also, during the execution, the external signals TS, CPUS, EE, and EOP may have occurred.

In a few cases, the general description of indicators shows that a single cause should actuate two indicators. However, in the event of such an alert, only the indicator of higher priority is actuated. The order of priorities of the indicators during indicator actuation is the same as the order shown on the indicator list except that IF has priority over AD.
Examples are:

a. In ADD TO MEMORY, when the operand in storage is protected, only indicator DS will be actuated. Indicator DF will remain unchanged.

b. In SWAP, however, an encountered protected address will actuate both indicators DS and DF because each of the two attempted transmissions causes different alerts.

c. An effective branch address that is both invalid and protected will cause only the actuation of indicator AD. Indicator IF will remain unchanged because an instruction has not been fetched.

d. If the address of the location of the subject instruction of EXECUTE or EXECUTE INDIRECT AND COUNT is both protected and invalid, indicator IF will be actuated. Indicator AD will not be actuated as a result of this invalid address unless the IF mask bit is zero.

Five indicators may be actuated in an input-output status alert: EPGK, UK, EE, EOP, and CS. These concern the input-output units specified by the channel address. CS is actually an independent alert, but, to avoid the need for a second channel address register, it is set with the other indicators.

Program Resumption

The state of execution at the occurrence of an alert, as known by the program in progress, should be saved to facilitate program resumption after action on the alert has been taken. For this reason, the execution of the instruction in progress is either carried through to completion or entirely suppressed before beginning action on the alert. When neither is possible, however, execution of the instruction is terminated.

The execution of the instruction during which an alert occurs is not affected by the state of the corresponding indicator mask bit except for alerts IF, DF, and XF. IF and DF alerts cause the suppression of the instruction in progress when their mask bits are one; otherwise they do not. An XF alert causes suppression of the instruction in progress only during index branching when the XF mask bit is one and the interruption system is enabled.

After the completion, suppression, or termination of an instruction execution, another instruction is fetched for execution. The location of this fetched instruction will be addressed by the instruction counter if interruption did not occur, or by the effective interruption address if interruption did occur.

Execution Completed

Generally, the execution of an instruction in progress will be completed in the event of an alert which:

1. Is not caused by the instruction in execution.
2. Is only a warning of a condition which is not necessarily an error.
3. Occurred because of the result of a completed execution.
4. Occurs with no loss of information; that is, the information is either directly available upon completion of the instruction or is recoverable by programming.

The indicators actuated in these cases are of class C (Figure 11).

PROGRAMMING NOTE

Indicator ZD, activated to indicate division by zero, is also included as Class C.

Execution Suppressed

Generally, the execution of an instruction in progress will be suppressed in the event of an alert which:

1. Indicates that the operation is not properly defined or cannot be executed because of the present state of the machine (OP, AD, EKJ, UNRJ, or CBJ).
2. Indicates that completion of the execution might destroy information (IJ, EXE, DS, DF, IF, or XF).
3. Indicates that the execution of EXECUTE, EXECUTE INDIRECT AND COUNT, or LOAD VALUE EFFECTIVE has exceeded one millisecond (the alert is USA). Any pseudo-instruction counters referred to by EXIC instructions will have been stepped.

The indicators actuated in these cases are of Class S (Figure 11).

In order to allow continuous computer operation, the suppression of an instruction must necessarily exclude the following operations:

1. The actuation of those indicators, associated with the instruction, of Class S, and indicators DF and IF when their mask bits are zero.
2. The actuation of indicators that were caused by conditions unrelated to the instruction in execution.
3. The stepping of the instruction counter.
4. The changing of the state of the interruption system which occurred as specification of BRANCH ENABLED, BRANCH DISABLED, or BRANCH ENABLED AND WAIT.

For example, an invalid effective bit address in BRANCH ON BIT causes the actuation of indicator AD and the suppression of the instruction, since no bit can be tested to complete its execution. Indicator IF cannot be actuated.

Another example is that, since a protected effective bit address in BRANCH ON BIT could result in the destruction of information, alert DS or DF may occur if:

1. The addressed bit is not specified to remain unconditionally unchanged; that is, bits 60 and 61 of the instruction are not both zero. The instruction is suppressed without a test of the addressed bit, and indicator DS is actuated.
2. The addressed bit is specified to remain unconditionally unchanged, and the DF mask bit is one. The instruction is suppressed without a test of the addressed bit, and indicator DF is actuated.
3. The addressed bit is specified to remain unconditionally unchanged, and the DF mask bit is zero. Indicator DF is actuated but does not cause suppression of the instruction.

A third example is that, when the instruction counter contents are stepped in normal fashion beyond the limit of storage addresses, the instruction counter addresses a void location and indicator AD is actuated. If the address is protected, indicator IF is actuated, instead. The instruction is suppressed.

Since an instruction is fully interpreted before it can be suppressed, more than one alert of Class S can be caused by the instruction. For example, if a STORE INSTRUCTION COUNTER IF COUNT, BRANCH AND REFILL is executed as the subject instruction of EXECUTE, it is possible for (1) the condition for branching to be met, (2) the effective SIC address to be invalid, (3) the effective branch address and the refill address to be protected, and (4) the count to reach zero. An attempt to execute this instruction causes the entire subject instruction to be suppressed and the actuation of all the indicators EXE, AD, IF, and DF. The protected effective branch address and the invalid effective SIC address are encountered because the condition for branching is met. When, however, in this example, the condition for branching is not met, the entire subject instruction is still suppressed but only indicator DF is actuated (the DF mask bit is assumed to be one). Indicator DF is actuated since the test for the refilling is independent of the test for a successful branch.

The occurrence of an alert of Class S during a transmission in TRANSMIT or SWAP or during the execution of a subject instruction of EXECUTE or EXECUTE INDIRECT AND COUNT may not cause full suppression of the instruction.

Execution Terminated

The execution of the instruction in progress will be terminated when one of these three types of alert occurs:

1. The alert indicates that a machine malfunction occurred during the instruction in progress in such a way that suppression or completion of the execution will not eliminate destruction of information. The indicators actuated in this case are of Class H (Figure 11).

2. The alert is of Class S but has occurred during a transmission in TRANSMIT or SWAP or during the execution of a subject instruction in EXECUTE or EXECUTE INDIRECT AND COUNT. Such an alert has the following effect:

   a. An alert of Class S during a transmission causes suppression of that transmission and all that follow; prior transmissions cannot be suppressed.

b. An alert of Class S during an execution operation causes suppression of the entire execution instruction except that (1) every pseudo-instruction counter that has been stepped prior to the alert remains stepped, and (2) any transmissions of SWAP or TRANSMIT, as subject instructions, which occurred before the alert are not suppressed.

3. An alert of Class S during progressive indexing suppresses only the indexing portion of the operation.

EXAMPLES OF TERMINATION

1. A protected effective transmission address in SWAP will cause the actuation of indicators DS and DF, the suppression of both the data fetch and the data store at that location, and suppression of all transmissions that follow. The transmissions that have already been completed cannot be suppressed.

2. A protected effective subject instruction address of EXECUTE INDIRECT AND COUNT will cause the actuation of indicator IF and the suppression of the instruction except for the stepping of the pseudo-instruction counter that has already been stepped.

3. A protected effective pseudo-instruction counter address in EXECUTE INDIRECT AND COUNT will cause actuation of indicator DS and suppression of the entire instruction. The pseudo-instruction counter will not be stepped.

4. If EXECUTE INDIRECT AND COUNT is the subject instruction of EXECUTE INDIRECT AND COUNT, a protected effective subject instruction address of the second-level execution instruction will cause suppression of the second-level execution instruction, but both pseudo-instruction counters will remain stepped.

5. If SWAP is the subject instruction of EXECUTE INDIRECT AND COUNT, an invalid transmission address will cause the actuation of indicator AD and the suppression of that transmission and all which follow; the pseudo-instruction counter will remain stepped and all transmissions that occurred prior to the alert will not be suppressed.

System States

The four system states are: running, programmed waiting, initial program loading, and initial power on. The computer system is designed for continuous running, but the other three states of the system are necessary. The programmed wait is accomplished by executing BRANCH ENABLED AND WAIT.

Two status lights are provided with the system to help indicate the system's state of sequential operation.

Initial Program Loading

The INITIAL PROGRAM LOAD key may be used for program loading and for manual termination of computer operation. A depression of this key has the following effect on the central processing unit:

1. The current operation is terminated.
2. The interruption system is disabled.

When the operation of the computer is terminated by means of the INITIAL PROGRAM LOAD key, all storage locations and registers that were altered prior to the termination are not restored to a former nor any other predetermined state, and hence the original program cannot normally be resumed. No further instructions are executed until a channel signal is received.

The INITIAL PROGRAM LOAD key is physically located on the operator's console, although not an integral logical part of it. If the system does not contain a console, the load key is provided in a separate box which may be placed at a convenient operating point.

PROGRAMMING NOTE

Since depression of the INITIAL PROGRAM LOAD key causes termination of computer operation, changes in the instruction counter as well as changes in storage may have occurred because of the interpretation of more than one instruction in preparation for execution. Such changes would make resumption of the terminated program at a later time undesirable.

Initial Power On

When the power is first turned on, the computer is in the following state:

1. The instruction counter and registers 0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, and 12 contain random bits not necessarily with proper parity.
2. The interruption system is disabled.
3. All control units are restored to their initial power-on status.

After the power is turned on, no instructions are executed until the computer has been signalled. The INITIAL PROGRAM LOAD key may be depressed for this purpose.

PROGRAMMING NOTE

The first program executed after the power is turned on may be a routine designed to alter the contents of registers 0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, and 12, since a fetch from one of these without proper parity may cause an alarm.

System Status Lights

Two lights are provided with the system to show the status of sequential operation.

1. Running Light. This light is turned on when the power is on and the system is not in a programmed wait status, initial program load status, or initial power-on status.

2. Inactive Light. This light is turned on when the power is on but the instruction counter is not being incremented.

Thus, the four possible states of these two lights are interpreted as follows:

INACTIVE	RUNNING	
Off	Off	Power off.
Off	On	Running normally.
On	Off	Programmed wait, initial program load, or initial power on.
On	On	Trouble: the system is hung up by continuous interruption, or lost control signal.

These status lights are located on the same unit as the INITIAL PROGRAM LOAD key.

Variable Field-Length Data Handling

The variable field-length data handling operations may be divided into three general classes: integer arithmetic operations, radix conversion operations, and connective operations. All of these three types operate on a variable-length data field, which may start at any bit position in storage. The instruction contains a 24-bit word and bit address that defines the leftmost bit of the storage field. The instruction format also includes a field that specifies the length in bits of the storage field. Another field in the instruction specifies the *byte size* of the addressed operand. The term "byte" describes a group of bits that represent a digit, alphabetic character, or other single symbol. Data fields in storage may be composed of bytes of any size from 1 to 8 bits.

The second operand in most of these operations is a field from the accumulator. Since this field may be located anywhere in the 128-bit accumulator, the instruction format contains an offset field that defines the low-order bit of the accumulator operand. The low-order positions of the two operands are aligned for processing. In some operations, the result replaces the accumulator operand; in others, the result replaces the storage operand; in yet others only certain indicators are affected, with both operands remaining unchanged.

Integer Arithmetic Operations

This class includes the elementary arithmetic operations. The variable-length data may be signed or unsigned, decimal or binary. When the data are signed, flag bits may be associated with them. The name "integer arithmetic" is used because the results of multiplications and divisions are aligned as if the operands were integers. The elementary arithmetic operations are:

+	Add
L	Load
ST	Store
*	Multiply
/	Divide

A set of modifiers is provided to alter the operand from storage so that these five operations effectively include a great many others. One modifier changes ADD to an algebraic subtraction; therefore, no separate SUBTRACT operation is provided. Another modifier permits integer arithmetic on unsigned numbers. A third modifier specifies the choice of binary or decimal arithmetic. Decimal multiplication and division are not provided directly in the machine, but their operation codes are provided. These may cause automatic entry to a subroutine, where the desired operation may be rapidly and easily performed using the conversion operations and binary multiplication or division.

Besides the modifications that are specified by the modifiers, other variations of the basic operations are provided. Variations on the elementary addition operation are:

+ MG	Add to Magnitude
M +	Add to Memory
M + MG	Add Magnitude to Memory
M + 1	Add One to Memory

The last three operations above add a quantity to storage rather than to the accumulator. The two magnitude operations provide for different treatment of the accumulator sign and inhibit sign reversal. Instead of overdraws they provide a zero result.

The comparison operations provide a comparison between two operands resulting in a low-equal-high test; no numeric result is produced, and both operands remain unchanged. Six operations are provided, the last four permitting successive comparisons to be followed by a single test.

K	Compare
KF	Compare Field
KE	Compare if Equal
KFE	Compare Field if Equal
KR	Compare for Range
KFR	Compare Field for Range

The operations COMPARE, COMPARE IF EQUAL, and COMPARE FOR RANGE provide a signed comparison between the storage operand and the entire accumulator contents to the left of the offset. The operations COMPARE FIELD, COMPARE FIELD IF EQUAL, and COMPARE FIELD FOR RANGE ignore the accumulator sign and provide a comparison between fields of equal length in digits, either binary or decimal.

One variation on the LOAD operation is LOAD WITH FLAG that loads the data flags into the accumulator sign byte register in addition to performing all the functions of LOAD.

One variation on STORE combines rounding with the store operation:

SRD	Store Rounded

The operation LOAD TRANSIT AND SET loads the transit register and provides automatic entry to a subroutine. It may be used, for example, to provide a square root operation or any other desired special function.

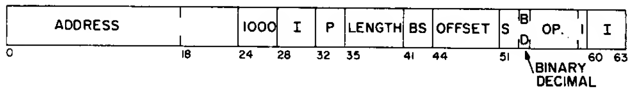
A variation on the multiply operation and a related auxiliary function are provided by

	LFT	Load Factor
and	* +	Multiply and Add.

The set of operations that is available for integer arithmetic is paralleled by a set of operations that perform arithmetic in a binary floating point mode. The floating point operations assume a special data format that allows these operations to be fully specified in a half-word instruction format. In addition to the set of operations that parallels the integer set, special floating point operations are also available that simplify double-precision arithmetic and facilitate separate handling of exponents and fractions.

Format

The integer arithmetic operations use the full-word instruction format shown below.



Bits 0-23 may be used as a word and bit address. This field is indexed by the contents of the index register specified in the I field of the first half-word, bits 28 through 31. The effective address may be used to designate the leftmost bit of a storage field, or it may be used as data. The three high-order bits of the second half-word form the P field of the instruction and specify the choice between direct and immediate addressing and the method of indexing to be used in modifying the first half-word of the instruction. Although the second half-word may be indexed, the P field is not modified. The index field I in bits 60 through 63 specifies the index word that is used to modify bits 35 through 50 of the instruction. On indexing the second half-word, bit position 3 of the index word is aligned with bit position 35 of the instruction, the three high-order bits of the index word being ignored. If the same index register is specified in both halves of an instruction while using progressive indexing, the value of the index register before modification is used in indexing the second half of the instruction.

The *field length,* bits 35 through 40, defines the length of the operand from storage, and the length of the result field when the result replaces the storage operand. The *offset field,* bits 44 through 50, specifies the number of bits by which the right end of the accumulator operand is offset from the right end of the accumulator. The byte size field, bits 41 through 43, specifies the *byte size* of the storage operand.

The operation code designates the operation to be performed. All integer arithmetic operations are subject to three *modifiers* specified in bit positions 51 through 53 of the instruction. One modifier, bit 51, designates whether the storage field is signed or not; another, bit 52, designates whether the sign of the unreplaced field is to be taken as-is or inverted; the third, bit 53, designates whether the operation is decimal or binary.

Field Definition

The operand specified explicitly by an instruction is defined by a word address, bit address and field length. If direct addressing is specified, the indexed word-and-bit addresses are used to define the leftmost bit of a storage field. The field length specifies the total length in bits of the storage operand, including the sign byte when present. A length field of zero in an instruction specifies a 64-bit storage field.

The storage word or word pair which contains the addressed memory field is brought from storage to the arithmetic section of the computer. There, the desired field is extracted, starting with the right end of the field. Processing is done from right to left, even though the bit address in the instruction specifies the leftmost bit of the field.

In integer arithmetic operations that affect storage, such as STORE and ADD TO MEMORY, the result replaces the storage operand in the bit positions specified by the address and field length. Data to the right or left of the specified field in storage are never affected.

If immediate addressing is specified, the effective address itself is used as the operand. The field length specifies the number of bits in the field starting with the leftmost bit of the effective address. The sign bit of the effective address is not used. If the field length designated is greater than 24, zeros are added to the right to obtain a field of proper length. The resulting field is treated in every respect like a field from storage. Immediate addressing is, of course, not possible in operations whose result is placed in storage. If immediate addressing is specified in a to-memory operation, the operation is not performed and the operation code invalid (OP) indicator is turned on.

The implied second operand of most operations is a field from the accumulator. The right end of the accumulator operand is defined by the offset specified in the instruction. The left end is the left end of the accumulator. For example, any carries that occur in an accumulator-altering operation may propagate to the left end of the accumulator. If a carry attempts to propagate beyond the left end of the accumulator, the carry is lost and the lost carry indicator is turned on.

It should be pointed out that the entire contents of the accumulator are treated basically as a unit. This means that all bits in the accumulator, even those to the right of the offset, can be changed as a result of an accumulator-altering operation. This occurs when such an operation causes the accumulator to be cleared or recomplemented. For example, LOAD clears the entire accumulator regardless of the offset specified. Recomplementation occurs when the sign of the accumulator changes as a result of an operation. In this event, the entire accumulator is recomplemented regardless of the offset.

In accumulator-altering operations, the offset can be conveniently thought of as equivalent to specifying the number of numeric zero bits to be added to the low-order end of the other operand before it is combined with the accumulator contents. In storage-altering operations, the contents of the accumulator to the right of the offset are ignored.

In decimal operations, the accumulator is treated as 32 four-bit digit positions. Decimal fields may occupy any set of contiguous digit positions in the accumulator. In decimal operations, the offset is limited to these digit positions and hence to a multiple of four by ignoring the two low-order bits of the offset specified by the effective address of the second half-word.

Programming Note

Since the accumulator is addressable, it may also be used as the storage operand of any of the data handling operations. The left half of the accumulator has address 8, the right half has address 9, and the accumulator sign byte register has address 10. The accumulator contents can be used as both operands of an operation. For example, the accumulator contents can be added to or subtracted from itself in an unsigned operation.

When the accumulator is used as the addressed operand in a signed operation, the low-order bits of the addressed accumulator field are used as the sign byte as in any addressed storage operand. The contents of the sign byte register are not used as the sign and flag bits of the addressed accumulator operand unless the addressed field includes bits 0-7 of word 10 as the low-order bits of the field. The sign and flag bits in the sign byte register apply to the accumulator contents only when the accumulator is the implied operand of an operation.

Sign Control

All the integer arithmetic operations include two modifiers, bits 51 and 52, which affect the use of the sign of one or both of the operands.

Numeric data fields in storage may be signed or unsigned. If the field is signed, the rightmost byte is the sign byte. The size of the sign byte is specified by the byte size. The sign of the accumulator operand is contained in the fifth bit position from the left in the accumulator sign byte register.

Instruction bit 51, the *unsigned* modifier, designates whether the addressed storage operand contains a sign byte or not. If the bit is zero, the operand is signed. If the bit is one, the operand is unsigned. A positive sign is assumed for an unsigned field and all bytes including the rightmost byte are processed as data.

Instruction bit 52, the *negative sign* modifier, designates whether the sign of the unreplaced operand is taken as-is or inverted before processing takes place. If the bit is zero, the same sign is used; if the bit is one, the sign is inverted. In general, if the result of an operation is placed in the accumulator, bit 52 determines the use of the sign of the operand from storage; if the result is placed in storage, bit 52 determines the use of the accumulator sign. The negative sign modifier, when one, has the effect of changing algebraic additions to subtractions or changing the sign of the result of multiplications, divisions, or stores.

The following table summarizes the combinations which can be specified.

BITS	TO-ACCUMULATOR OPERATION		TO-MEMORY OPERATION	
51-52	ACC. SIGN	STORAGE FIELD SIGN	ACC. SIGN	STORAGE FIELD SIGN
00	Use as-is	Use as-is	Use as-is	Use as-is
01	Use as-is	Invert	Invert	Use as-is
10	Use as-is	Unsigned, use +	Use as-is	Unsigned, use +
11	Use as-is	Unsigned, use —	Invert	Unsigned, use +

Data Flag and Zone Bits

In some applications it may be desirable to mark certain data to indicate that special handling of that data field is required. The data flag bits in the sign byte of signed data fields allow such marking.

All integer arithmetic operations that obtain data flag bits from storage place these bits into the indicator register. LOAD WITH FLAG also places the flag bits into the three rightmost bits of the accumulator sign byte register. The only other integer arithmetic operation that affects the data flag positions of the sign byte register as an inherent part of its operation is LOAD, which sets these three bit positions to zero.

Zone bits in the sign byte of a signed storage operand do not replace the zone bit positions in the sign byte register as an inherent part of any operation. Both the zone bit and flag bit positions of the sign byte register may be changed by addressing them as the storage operand of a to-memory operation.

Operations that place a signed result in storage, with the exception of STORE and STORE ROUNDED, do not alter the original flag bits or zone bits of the storage operand sign byte. STORE and STORE ROUNDED replace the data flag and zone bits in the sign byte and the zone bits in decimal bytes with the contents of the corresponding bit positions in the accumulator sign byte register.

Radix and Byte Size

Bit 53 of the instruction, the *radix* modifier, specifies the type of arithmetic to be performed, the interpretation of the byte size field in the instruction, and the byte size of the accumulator operand. Binary arithmetic is specified if the bit is zero; decimal arithmetic is specified if the bit is one.

In a binary operation, the byte size field, bits 41-43, of the instruction defines only the length of the sign byte since there is no inherent subdivision among the numeric bits of a binary number. The remainder of the field is processed in eight-bit bytes whenever possible, with the exception of the last byte whose size is the number of bits necessary to complete processing of the specified field length. In signed binary operations to storage, the stored result includes a sign byte of the size specified.

If decimal arithmetic is specified, the byte size of the instruction defines the length of each byte in the storage field including the sign byte if the data are signed. Decimal digits are represented in four-bit binary-coded decimal form and, therefore, can be stored in compact form. The number of bits used to represent a digit in storage can be more than four bits, however; for example, two zone bits may be added to allow decimal information to be readily interspersed with alphabetic data coded in six-bit form. Decimal arithmetic can be performed on data of any byte size. If the byte size is greater than four, only the four low-order bits of each byte take part in the operation. The remaining bits of each byte are ignored. If a byte size less than four is specified in a decimal operation, normal decimal results may not be produced if two operands are involved whose effective signs are unlike, owing to the complementing procedure used. When complementation is not involved, each byte of the storage operand is converted to a four-bit byte for processing by the addition of high-order zeros. The byte size of the accumulator operand is assumed to be four in all decimal operations. The accumulator is always composed of integral 4-bit bytes regardless of the byte size or field length of the storage operand.

Since the field lengths are not constrained to be multiples of the byte size, the high-order byte of the memory operand may have an effective byte size less than that specified in the operation. The high-order byte is processed according to its effective byte size rather than the byte size specified in the operation.

If the field length of a signed operation is less than the byte size, the byte size is used to determine the format of the sign byte so that the bits specified are properly used. For example, a LOAD operation with byte size four and field length two treats the two addressed bits as the U and V flags and sets the corresponding positions of the indicator register accordingly. The accumulator and the four low-order bits of sign byte register are cleared, since there is no sign or data in the addressed field.

In decimal operations which alter storage, the result byte generated by processing a byte from the storage field and a byte from the accumulator replaces the byte which was obtained from the storage field. The replacement is made in accordance with the byte size of the storage field. If the specified byte size is greater than four, the four bits making up each result digit replace the four low-order bits of each byte of the storage field. In STORE and STORE ROUNDED, the zone bits of each byte in the storage field are replaced by the contents of the corresponding zone positions in the accumulator sign byte register; they remain unchanged in all other arithmetic operations to storage. If the specified byte size is less than four, each byte of the storage field is replaced by the number of bits specified by the byte size. These bits are the low-order bits of each result byte. Higher order bits in each result byte are dropped, and no indication is given if these positions contain one-bits. If the high-order byte of the addressed storage field is less than the byte size, it is replaced by the low-order bits of the corresponding result byte. Higher order bits are dropped, and no indication is given if they are non-zero.

Indicators

Several indicators in the indicator register are set as a result of the integer arithmetic operations. These include the general result exception, the data flag, the transit, and the arithmetic result indicators. The general result exception and the transit indicators are permanent. If they are turned on by any operation, they will remain on until they are turned off either by causing an interruption or by being explicitly made zero by an operation that addresses them as an operand. For example, BRANCH ON INDICATOR or CONNECT TO MEMORY can accomplish this function. The data flag and arithmetic result indicators are temporary. At the completion of an integer arithmetic operation, all indicators of this type that apply to the particular operation will be on if the necessary condition arose during the operation; otherwise, they will be off. They then remain set until another operation which affects them is executed.

The indicators that are set as a result of the integer arithmetic operations are described below.

Lost Carry (LC). In to-accumulator operations, this indicator is turned on when a carry attempts to propagate beyond the left end of the accumulator. In binary to-memory operations, this indicator is actuated by a carry beyond the leftmost bit of the addressed field. In decimal to-memory operations, this indicator is actuated by a decimal carry beyond the leftmost decimal digit of the addressed field. Thus, if the high-order byte of a decimal storage operand is less than four bits in length, the lost carry indicator is actuated only if the corresponding result byte from the processing unit is greater than nine. The high-order bits that are dropped in truncating the high-order byte for storage cannot actuate this indicator even if they are non-zero. This indicator is also actuated when a to-memory operation with an unsigned storage operand produces a negative result. In all cases, the operation is completed normally except for the loss of the carry.

Partial Field (PF). In to-accumulator operations, this indicator is turned on when significant bits of the storage operand overlap the left end of the accumulator. The accumulator operand is extended with high-order zeros to combine with the storage operand, but these high-order positions are dropped from the result. Otherwise, the operation is completed normally. In to-memory operations, the indicator is turned on when there are non-zero accumulator positions to the left of the accumulator field that is participating in the operation. In binary operations to storage, this indicator is turned on if there is any one-bit in the accumulator to the left of the participating field. In decimal operations to storage, the indicator is turned on if there is any non-zero digit to the left of the highest order accumulator digit that is taking part in the operation. Partial field is also turned on if a binary multiplier, multiplicand, or divisor exceeds 48 significant bits in length or if a binary dividend exceeds 96 significant bits.

Zero Divisor (ZD). This indicator is turned on when the divisor in a DIVIDE operation is zero. The division is not performed.

Data Flag T (TF); Data Flag U (UF); Data Flag V (VF). These indicators are set according to the data flag bits, if any, of the storage operand. Any operation that normally affects these indicators will turn them off if the corresponding data flags are not obtained from storage.

Binary Transit (BTR). This indicator is turned on by any binary LOAD TRANSIT AND SET operation.

Decimal Transit (DTR). This indicator is turned on by any decimal LOAD TRANSIT AND SET, MULTIPLY, MULTIPLY AND ADD, or DIVIDE operation.

To-Memory Operation (MOP). This indicator is turned on by all integer arithmetic operations in which the result is placed in the addressed storage location. All other integer arithmetic operations turn this indicator off.

Result Less than Zero (RLZ); Result Zero (RZ); Result Greater than Zero (RGZ); Result Negative (RN). These arithmetic result indicators are set according to the result of all integer arithmetic operations except the comparison operations.

Accumulator Low (AL); Accumulator Equal (AE); Accumulator High (AH). These indicators are set according to the result of the comparison operations. They refer to the accumulator operand as compared with the storage operand.

Operations

It is useful to classify the twenty integer arithmetic operations into categories two different ways. The classifications are shown in the list below.

		RESULT ALTERS	ARITHMETIC ACTION
+	Add	Accumulator	Add
+MG	Add to Magnitude	Accumulator	Add
L	Load	Accumulator	Replace
LWF	Load with Flag	Accumulator	Replace
M+	Add to Memory	Storage	Add
M+MG	Add Magnitude to Memory	Storage	Add
M+1	Add One to Memory	Storage	Add
ST	Store	Storage	Replace
SRD	Store Rounded	Storage	Replace
K	Compare	Indicators only	Add
KF	Compare Field	Indicators only	Add
KE	Compare if Equal	Indicators only	Add
KFE	Compare Field if Equal	Indicators only	Add
KR	Compare for Range	Indicators only	Add
KFR	Compare Field for Range	Indicators only	Add
LTS	Load Transit and Set	Transit	Replace
*	Multiply	Accumulator	Multiply
LFT	Load Factor	Factor	Replace
*+	Multiply and Add	Accumulator	Multiply and add
/	Divide	Accumulator and remainder	Divide

Add (+)

The addressed operand is algebraically added to the contents of the accumulator with specified offset. The sign bit in the accumulator sign byte register is set in accordance with the sign of the algebraic sum. The remaining bits in the sign byte register remain unchanged.

Modifiers

Unsigned. This modifier describes the addressed operand.

Negative Sign. This modifier designates whether the sign of the storage operand is taken as-is or is inverted before the operation takes place. Unsigned fields are considered positive before this modifier is applied.

Radix. This modifier specifies the accumulator byte size, determines the properties of the adder, and controls complementation and recomplementation when the effective operand signs are unlike. In decimal arithmetic, four-bit bytes are used, only the four low-order bits of each byte in the addressed operand participate in the operation, and a carry occurs from one byte to the next when a sum byte exceeds 9. In decimal operations, the two low-order bits of the effective offset are ignored and considered zero. Thus, the offset is always a multiple of four. In binary arithmetic, an eight-bit byte is used whenever possible. Any offset may be specified in a binary operation.

Indicators

Lost Carry (LC). This indicator is actuated if information is lost because a carry attempts to propagate beyond the left end of the accumulator.

Partial Field (PF). This indicator is turned on if significant bits of the data field from storage overlap the left end of the accumulator. The accumulator operand is extended with high-order zeros to combine with the storage operand, but the high-order bits of the result are dropped. A partial field condition will not cause a lost carry indication. However, both conditions may arise during the execution of a single instruction.

Data Flags (TF, UF, VF). These indicators are set according to the flag bits of the storage operand.

To-Memory Operation (MOP). This indicator is turned off.

Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set according to the contents of the entire accumulator when the operation is complete.

Programming Notes

The sign modifiers apply to the storage operand in the ADD operation. Note that using the negative sign modifier causes an algebraic subtraction, whether the field addressed is signed or unsigned. There is therefore no need for a subtract operation, and no separate operation is provided. Furthermore, using the unsigned modifier is equivalent to absolute value addition or subtraction; there is no sign on the addressed operand so a plus sign is assumed. The accumulator sign is not ignored, but influences the operation in the usual manner. It is important to realize that all add-type operations operate upon the contents of the entire accumulator. If an addition of operands with unlike signs is performed, there is a possibility of sign change. When this occurs, the whole contents of the accumulator may be changed, including the part below the offset. Consider the examples:

		INDICATOR
1. ADD, L (376+) offset 8 bits	75679+	
(2 bytes) where acc. contains	37600+	
75679+	113279+	Result greater than zero
2. ADD, L (532−) offset 8 bits	45623+	
where acc. contains 45623+	53200−	
	7577−	Result less than zero Result negative
3. ADD, L (532−) offset 8 bits	53200+	
where acc. contains 53200+	53200−	
	00000+	Result zero
4. ADD, L (532+) offset 8 bits	53200−	
where acc. contains 53200−	53200+	
	00000−	Result zero Result negative
5. ADD, L (532−) offset 8 bits	53201+	
where acc. contains 53201+	53200−	
	00001+	Result greater than zero

The result in every case is the same as one would get performing the same operations by hand if one considers the offset as describing the number of low-order zeros that are to be attached to the addressed operand. On any operation in which the result in the accumulator is all zero, the sign remains as it was before the operation began.

Add to Magnitude (+ MG)

The addressed operand is algebraically added to the magnitude of the contents of the accumulator with specified offset. The accumulator operand is assumed positive for purposes of the addition itself. If the result is positive, the sum is placed into the accumulator. If the result is negative, the entire accumulator is cleared to zero. In either case, the content of the accumulator sign byte register is not changed.

MODIFIERS

Unsigned; Negative Sign; Radix. These modifiers operate as in ADD.

INDICATORS

Lost Carry (LC); Partial Field (PF); Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set as in ADD.

PROGRAMMING EXAMPLE

Figure 12 shows a simple use of ADD TO MAGNITUDE. Given two 64-bit unsigned binary numbers *x* and *y*, put the larger of the two in the accumulator.

NAME	STATEMENT	NOTES
	L (BU, 64, 8), X, 0	
	- MG (BU, 64, 8), Y, 0	1
	+ (BU, 64, 8), Y, 0	2

Notes:		If x⩾y	If x<y
1.	Result is	x-y	0
2.	Result is	x	y

Figure 12. Program Example, Add to Magnitude

Load (L)

The entire left and right halves of the accumulator and the four low-order bits of the accumulator sign byte register are cleared. The numeric part of the addressed operand is placed in the accumulator as specified by the offset. The sign bit in the sign byte register is set according to the sign of the data as affected by the sign modifiers. The zone bits of the accumulator sign byte are not altered.

MODIFIERS

Unsigned; Negative Sign; Radix. These modifiers operate as in ADD.

INDICATORS

Partial Field (PF); Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set as in ADD.

Load with Flag (LWF)

The LOAD WITH FLAG operation is the same as LOAD except that if the addressed storage field includes data flag bits, these bits are placed in the accumulator sign byte register in addition to setting the data flag indicators.

MODIFIERS

Unsigned; Negative Sign; Radix. These modifiers operate as in ADD.

INDICATORS

Partial Field (PF); Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set as in ADD.

PROGRAMMING NOTE

LOAD, LOAD WITH FLAG, and LOAD CONVERTED are the only variable-field-length data handling operations that affect the data flag bit positions in the accumulator sign byte register as an implied operand. All other operations of this type leave these positions unchanged. No operation changes the zone bit positions in the accumulator sign byte register as an implied operand.

Add to Memory (M +)

The accumulator field specified by the offset is algebraically added to the addressed operand. The accumulator sign is used as modified by the negative sign modifier, and the sum replaces the addressed field in storage.

MODIFIERS

Unsigned. This modifier describes the addressed field in storage. If the operation is signed, the sign bit of the storage operand is set to the sign of the algebraic sum.

Negative Sign. This modifier applies to the accumulator sign as used in the operation. The original accumulator sign remains unchanged in the sign byte register.

Radix. In a binary ADD TO MEMORY, the byte size refers only to the length of the sign byte, thereby determining the position of the sign bit. The remainder of the field is processed in eight-bit bytes whenever possible.

In a decimal operation, the field is processed in four-bit bytes. If the byte size of a decimal storage field participating in ADD TO MEMORY is greater than four, only the four low-order bits of each byte participate in the addition. The high-order bits of each byte are not changed by the operation. If a byte size less than four is specified in a decimal operation, each byte from the storage field is converted to a four-bit byte for processing by the addition of high-order zeros. Normal decimal results may not be produced if two operands are involved whose effective signs are unlike. Only the low-order bits of each sum byte replace the byte from storage, so that the stored result has the same byte size as the original storage field. Since normal decimal addition takes place, the high-order positions that are dropped from each sum byte may contain one bits. These are lost and no indication of the loss is given.

Indicators

Lost Carry (LC). This indicator is actuated by a carry beyond the leftmost digit, either binary or decimal, of the addressed field. This indicator is also turned on if the result of an unsigned operation is negative. In such a case, the magnitude of the result is stored in storage.

Partial Field (PF). This indicator is turned on if there are non-zero digits, either binary or decimal, in the accumulator to the left of the highest order digit that is participating in the operation.

Data Flags (TF, UF, VF). These indicators are set according to the data flag bits of the storage operand.

To-Memory Operation (MOP). This indicator is turned on.

Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set according to the algebraic sum of the two operands, whether the sign is stored or not.

Programming Notes

The data flag bits of a signed operand in storage are not altered by ADD TO MEMORY. The accumulator contents are not altered by ADD TO MEMORY.

If the negative sign modifier is one, ADD TO MEMORY performs an algebraic subtraction of the accumulator contents from the addressed field. Although the negative sign modifier applies to the accumulator sign in to-memory operations, the unsigned modifier always applies to the operand in storage. Unsigned storage fields are considered positive.

In operations to storage, the length of the accumulator field between the specified offset and the left end of the accumulator can be less than the number of bits from the storage operand which are to take part in the operation. In this event the accumulator operand is lengthened for the operation by the addition of high-order zeros.

Add Magnitude to Memory (M+MG)

The magnitude of the accumulator field as specified by the offset is algebraically added to the addressed operand. A positive sign is assumed for the accumulator field before the negative sign modifier is applied. The actual accumulator sign is not used or changed by the operation. If the sum has the same sign as the sign of the storage operand, it replaces that operand in storage. If the sum is of opposite sign, zero bytes replace the numeric part of the storage field. In either case the sign byte, if any, of the storage operand is not changed.

Modifiers

Unsigned. This modifier operates as in ADD TO MEMORY.

Negative Sign. The accumulator sign is considered positive when the negative sign modifier is zero and negative when the modifier is one.

Radix. This modifier operates as in ADD TO MEMORY.

Indicators

Lost Carry (LC); Partial Field (PF); Data Flags (TF, UF, VF); To-Memory Operation (MOP). These indicators are set as in ADD TO MEMORY.

Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set according to the final result stored.

Programming Note

The four operations ADD, ADD TO MAGNITUDE, ADD TO MEMORY, and ADD MAGNITUDE TO MEMORY allow a flexible handling of the signs of the factors and the result. Not all possible combinations are provided, however, and in some cases the results may not be obvious. Figure 13 shows the storage or accumulator field that is produced by all possible combinations of operations, sign modifiers, operand signs, and relative magnitudes of operands.

The storage field is assumed to be signed in all cases. When the unsigned modifier is specified, the field length is adjusted accordingly and the absolute value of the storage field is used in the operation. However, in an unsigned to-memory operation, the original sign of the field remains in storage and is therefore shown as part of the result field.

Storage operand →	3+	5+	3+	5+	3-	5-	3-	5-
Accumulator operand →	5+	3+	5-	3-	5+	3+	5-	3-
↓Operation and Modifiers								
Add	8+	8+	2-	2+	2+	2-	8-	8-
Add, N	2+	2-	8-	8-	8+	8+	2-	2+
Add, U	8+	8+	2-	2+	8+	8+	2-	2+
Add, U, N	2+	2-	8-	8-	2+	2-	8-	8-
Add to Magnitude	8+	8+	8-	8-	2+	0+	2-	0-
Add to Magnitude, N	2+	0+	2-	0-	8+	8+	8-	8-
Add to Magnitude, U	8+	8+	8-	8-	8+	8+	8-	8-
Add to Magnitude, U, N	2+	0+	2-	0-	2+	0+	2-	0-
Add to Memory	8+	8+	2-	2+	2+	2-	8-	8-
Add to Memory, N	2-	2+	8+	8+	8-	8-	2+	2-
Add to Memory, U	8+	8+	2+*	2+	8-	8-	2-*	2-
Add to Memory, U, N	2+*	2+	8+	8+	2-*	2-	8-	8-
Add Magnitude to Memory	8+	8+	8+	8+	0-	2-	0-	2-
Add Magnitude to Memory, N	0+	2+	0+	2+	8-	8-	8-	8-
Add Magnitude to Memory, U	8+	8+	8+	8+	8-	8-	8-	8-
Add Magnitude to Memory, U, N	0+	2+	0+	2+	0-	2-	0-	2-

*The Lost Carry indicator is turned on

Figure 13. Sign Condition Results

Add One to Memory (M+1)

A plus or minus one is algebraically added to the addressed storage field.

Modifiers

Unsigned. This modifier operates as in ADD TO MEMORY.

Negative Sign. This modifier specifies the sign of the one to be added. If the modifier is zero, a plus one is algebraically added to the storage operand; if it is one, a minus one is added.

Radix. This modifier operates as in ADD TO MEMORY.

INDICATORS

Lost Carry (LC); Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators operate as in ADD TO MEMORY.

PROGRAMMING NOTES

ADD ONE TO MEMORY is identical to an ADD TO MEMORY in which the accumulator operand is +1, except that the partial field indicator is not affected. This operation can be used for ordinary adding or counting by ones in signed or unsigned fields. Counting by any power of the specified radix (2 or 10) can be performed in fields whose signs are known by using an unsigned operation with proper bit address.

The offset field in ADD ONE TO MEMORY has no meaning and is ignored.

Store (ST)

The addressed field in storage is replaced by the accumulator field specified by the offset. If a signed operation is specified, a sign byte of proper size is obtained from the accumulator sign byte register and, after modification by the negative sign modifier, this sign byte replaces the rightmost byte of the storage field. The accumulator and the sign byte register remain unchanged.

MODIFIERS

Unsigned. This modifier determines whether the stored field is signed or unsigned. When the field is signed, the contents of the accumulator sign byte register replace the rightmost byte of the addressed field. If the byte size is less than eight, only part of the contents of the sign byte register is stored. The number of bits in the sign byte stored is determined by the specified byte size.

Negative Sign. If this modifier is one, the sign bit from the accumulator sign byte register is inverted before being placed in the storage field. The contents of the accumulator sign byte register remain unchanged. If the unsigned modifier is specified, the negative sign modifier has no effect on the field stored. However, the result indicators are set according to the modified accumulator sign, whether it is stored or not.

Radix. In a binary STORE, the byte size refers only to the length of the sign byte stored. The specified accumulator data field is stored in 8-bit bytes wherever possible. In a decimal STORE, the byte size refers to the size of each byte in the storage field, including the sign byte if the operation is signed. Each 4-bit byte of the accumulator replaces the four low-order bits of each byte of the storage field except the sign byte. If the specified byte size is greater than four, the remaining high-order bits of each byte are replaced by the corresponding zone bits in the accumulator sign byte register. If the specified byte size is less than four, only the low-order bits from each accumulator byte replace each byte in the storage field.

INDICATORS

Partial Field (PF). The partial field indicator is turned on in a binary STORE if there is any one-bit in the accumulator to the left of the last bit position stored. In a decimal STORE, this indicator is turned on if there is a non-zero byte to the left of the high-order byte stored. If the field length is not a multiple of the byte size in a decimal STORE, only part of the high-order digit is stored so that only the addressed field is replaced in storage. If all accumulator bytes to the left of the entire high-order accumulator byte taking part in the operation are zero, the partial field indicator is not turned on even if the high-order bits of the high-order byte that were not stored are non-zero.

To-Memory Operation (MOP). This indicator is turned on.

Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set according to the data field stored. The sign of the accumulator, as modified by the negative sign modifier, affects the setting whether it is stored or not.

PROGRAMMING NOTES

The field length and offset specified in a STORE instruction may be such that the field to be stored extends beyond the left end of the accumulator. In this event, the accumulator operand is lengthened for the operation by the addition of high-order zeros. In a binary operation, this results in replacing the left end of the storage field with zeros. In a decimal operation, the high-order part of the storage field is replaced by bytes whose numeric part is zero and whose zones, if any, are those provided from the sign register.

A signed STORE operation always stores sign bytes with the data flag bits and zone bits that are in the accumulator sign byte register. To maintain the original flag bits on a data field which is to be processed and then stored, LOAD WITH FLAG should be used when the field is initially obtained. It should be remembered that LOAD clears the data flag positions in the sign byte register and does not place the flag bits of the addressed operand into these positions.

Desired zone bits must be placed into the zone bit positions of the sign byte register by explicitly addressing these positions as the operand of a to-memory operation. These positions will then remain unchanged until they are once again explicitly addressed in a to-memory operation.

Since data flow into storage in a decimal STORE is controlled by the byte size in the instruction, while data flow from the accumulator takes place in four-bit bytes, STORE can be used to store decimal data in any required byte size. This action is the reverse of a decimal LOAD in which only the four low-order bits of each storage byte are placed into the accumulator. This feature provides a convenient means of converting decimal data from one byte size to another. Thus, it is a simple programming operation to keep numerical files in space-conserving four-bit bytes and still receive input or prepare output in six- or eight-bit code without extra program steps for the conversion. The byte size conversion takes place automatically during processing of the data when the proper byte sizes are specified in the instructions used in the processing. Note, however, that any zone bits present with the decimal digits would be lost if data were converted in this fashion. The connective operations provide a more general means of converting data from one byte size to another.

The data flag indicators are not changed as a result of STORE. These indicators remain as set before the operation. The lost carry indicator cannot be turned on by a STORE operation.

PROGRAMMING EXAMPLE

Figure 14 shows the use of the variable-field-length features and sign modifiers. Let A and B be two binary numbers stored in consecutive positions in storage starting at location 1776.00. Each number contains sixteen numeric bits and a sign byte of eight bits. Form $A - B$, $|A - B|$, $-|A - B|$, and $\overline{A - B}$ and store in consecutive memory positions starting at location 1774.40. The notation $\overline{A}$ represents the unsigned quantity.

NAME	STATEMENT	NOTES
	L (B, 24, 8), 1776.0, 0	
	- (B, 24, 8), 1776.24, 0	
	ST (B, 24, 8), 1774.40, 0	1
	L (BU, 16, 8), 1774.40, 0	
	ST (B, 24, 8), 1775.0, 0	2
	STN (B, 24, 8), 1775.24, 0	3
	STN (BU, 16, 8), 1775.48, 0	4

Notes:
1. Stores A-B
2. Stores |A-B|
3. Stores - |A-B|
4. Stores $\overline{A-B}$. The same result would have been obtained if the negative sign modifier had not been used, except for the setting of the arithmetic result indicators.

Figure 14. Sign Modifier Example

Store Rounded (SRD)

The field from the accumulator is rounded by adding to its magnitude one-half of the proper radix in the position to the right of the offset. Carries are propagated as in any addition in that radix. This rounded field replaces the addressed storage field. If a signed operation is specified, the contents of the accumulator sign byte register replace the rightmost byte of the storage field after the sign bit has been modified by the negative sign modifier. The contents of the accumulator and the sign byte register are not changed by this operation. The accumulator sign is ignored in the actual rounding process.

MODIFIERS

Unsigned; Negative Sign. These modifiers operate as in STORE.

Radix. This modifier acts in the same manner as in STORE, except that in addition it governs the rounding process. If a binary operation is specified, a binary one is added to the field from the accumulator in the bit position immediately to the right of the specified offset. Carries are propagated as in any binary addition. If a decimal operation is specified, a four-bit byte with value five is added to the four-bit byte immediately to the right of the offset in the field from the accumulator. Carries are propagated as in any decimal addition. This action is equivalent to adding the five one decimal position to the right of the offset.

INDICATORS

Partial Field (PF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set as in STORE.

Lost Carry (LC). This indicator is turned on when the rounding process causes a carry to be propagated beyond the left end of the field stored.

PROGRAMMING NOTES

The STORE ROUNDED operation does not change the contents of the accumulator. This is convenient in those problems in which it is desirable to store part of the accumulator contents rounded while retaining full precision in the accumulator for further calculation. The accumulator contents can be rounded by using ADD TO MAGNITUDE with an immediate address of 1 or 5 with the proper offset. An unsigned STORE ROUNDED addressing the accumulator as the storage operand can also be used for this purpose.

A STORE ROUNDED with zero offset specified is equivalent to a STORE.

Compare (K)

The contents of the accumulator left of the specified offset are algebraically compared with the addressed operand. The bits to the right of the offset do not participate in the comparison. The comparison result indicators are set to describe the accumulator field as compared with the storage field. Neither operand is changed by the operation. The arithmetic result indicators are not affected.

Modifiers

Unsigned; Negative Sign; Radix. These modifiers operate as in ADD.

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP). These indicators are set as in ADD.

Accumulator Low (AL). This indicator is turned on if the accumulator operand is algebraically less than the storage operand.

Accumulator Equal (AE). This indicator is turned on if the accumulator operand is algebraically equal to the storage operand.

Accumulator High (AH). This indicator is turned on if the accumulator operand is algebraically greater than the storage operand.

Programming Notes

Comparison operates numerically and algebraically. Therefore:

1. If the magnitude of the two comparands are represented by x and y, where $x > y > 0$, then the comparison sequence of the comparands with sign can be obtained from the statement

$$+x > +y > +0 = -0 > -y > -x$$

In other words, if the signs of both operands are positive, the operand with greater magnitude is considered high. If the signs of the operands are unlike, the operand with positive sign is considered high and the operand with negative sign is considered low. If the signs of both operands are negative, the more negative operand, that with the greater magnitude, is considered low. A negative zero is considered equal to a positive zero.

2. Alphabetic comparison, including zone bits, requires that the binary modifier be used. An alphameric code can readily be constructed to give any desired collating sequence.

3. The comparison result indicators correspond to and substitute for the arithmetic result indicators in comparison operations. The arithmetic result indicators are not changed by a comparison operation.

4. The comparison is performed by making a subtraction and testing the result. A decimal digit with byte size less than four is handled as in any other subtraction.

If the field length and offset are such that high-order positions of the storage field compare with positions to the left of the left end of the accumulator, the accumulator field is extended by adding zeros to its left before the comparison is made.

Neither the lost carry nor the partial field indicator can be turned on by a comparison operation.

Compare Field (KF)

A field in the accumulator is algebraically compared with the addressed operand. This accumulator field is the same length in bytes as the field from storage exclusive of the sign byte, and its location is specified by the offset in the usual manner. The actual accumulator sign is not used by the operation. A positive sign is assumed for the accumulator field. The sign of the storage field is used as modified by the sign modifiers. The comparison result indicators are set to describe the accumulator field as compared with the storage field. Neither operand is changed by the operation. The arithmetic result indicators are not affected.

Modifiers

Unsigned, Negative Sign; Radix. These modifiers operate as in ADD.

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Comparison Result (AL, AE, AH). These indicators are set as in COMPARE.

Programming Notes

The field comparison operations are the only arithmetic operations that do not treat the entire accumulator contents as a single quantity. Comparisons are made between fields of equal length in numeric bytes. For this reason, no indication is given in these operations if there is a one-bit in the accumulator to the left of the field which is being compared with the storage operand.

Compare if Equal (KE)

If the accumulator-equal indicator is off when this operation is interpreted, the comparison is not performed and the comparison result indicators are not changed. Otherwise, the operation is executed exactly as is COMPARE.

Modifiers

Unsigned; Negative Sign; Radix. These modifiers operate as in ADD.

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP). These indicators are set as in COMPARE.

Comparison Result (AL, AE, AH). These indicators are set as in COMPARE if the accumulator-equal indicator is on when the operation is initiated. If the accumulator-equal indicator is off, these indicators are not affected by the operation.

Programming Note

COMPARE IF EQUAL can be used with COMPARE to obtain a comparison of fields of more than 64 bits. A COMPARE operation is used to compare the high-order part of both fields. This is followed by one or more COMPARE IF EQUAL operations comparing the succeeding lower-order parts of the field. At the completion of the series of comparisons, the indicators will describe the relation of the entire accumulator field to the entire storage field. The same technique may also be used to compare fields that are split into several parts.

Programming Example

Two programs for comparing two unsigned fields, M and N, each of 15 decimal digits in six-bit bytes, are shown in Figure 15. The first program, using COMPARE IF EQUAL, requires fewer instructions, but in some cases will require more execution time than the second program, which does not use this operation.

USING COMPARE IF EQUAL

NAME	STATEMENT	NOTES
	L (DU, 60, 6), M, 0	
	K (DU, 60, 6), N, 0	
	L (DU, 30, 6), M+ .60, 0	
	KE (DU, 30, 6), N+ .60, 0	

NOT USING COMPARE IF EQUAL

NAME	STATEMENT	NOTES
	L (DU, 60, 6), M, 0	
	K (DU, 60, 6), N, 0	
	BZAE, END	
	L (DU, 30, 6), M+ .60, 0	
	K (DU, 30, 6), N+ .60, 0	
END	Continuation of program	

Figure 15. Comparing Unsigned Fields

Compare Field if Equal (KFE)

If the accumulator-equal indicator is off when this operation is interpreted, the comparison is not performed and the comparison result indicators are not changed. Otherwise, the operation is executed exactly as is COMPARE FIELD.

Modifiers

Unsigned; Negative Sign; Radix. These modifiers operate as in ADD.

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Comparison Result (AL, AE, AH). These indicators are set as in COMPARE IF EQUAL.

Programming Note

COMPARE FIELD IF EQUAL can be used with COMPARE FIELD to obtain comparison of fields of more than 64 bits in the same way that COMPARE IF EQUAL is used with COMPARE.

Compare for Range (KR)

If the accumulator-high indicator is off when this operation is interpreted, the comparison is performed and the comparison result indicators are not changed. Otherwise, the operation is identical to COMPARE except for the setting of the comparison result indicators. If the accumulator operand is equal to or higher than the storage operand, the accumulator-high indicator is left on. If the accumulator operand is less than the storage operand, the AE indicator is turned on, and AH is turned off.

Modifiers

Unsigned; Negative Sign; Radix. These modifiers operate as in ADD.

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP). Set as in COMPARE.

Accumulator Equal (AE). Set if accumulator operand is algebraically less than storage operand.

Accumulator High (AH). Remains on if accumulator operand is algebraically equal to or greater than storage operand. Set to zero when accumulator operand is algebraically less than storage operand.

Programming Note

COMPARE FOR RANGE can be used with COMPARE to determine whether a quantity falls within a given range. The range is defined by two bounds stored in storage and includes all values equal to or greater than the lower bound but less than the upper bound. The field to be tested is placed in the accumulator. A COMPARE is then performed, addressing the lower range bound, followed by a COMPARE FOR RANGE, addressing the upper bound. If AE indicator is on at completion, the tested field is within range. If AL indicator is on, the field is below range; if AH is on the field is above range (Figure 16).

If a COMPARE FOR RANGE is executed with one or more AL, AE, or AH indicators on, more than one of these may be left on after execution of the COMPARE FOR RANGE. The COMPARE always causes one and only one indicator to be turned on. Therefore, if a COMPARE FOR RANGE follows a COMPARE, the above case never exists and indicators are set as described.

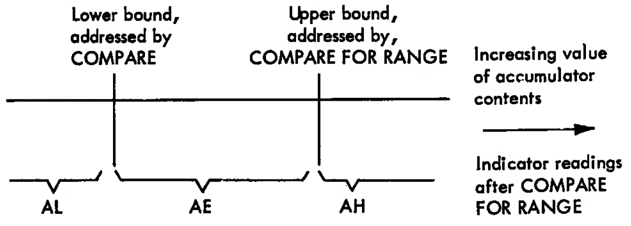

Lower bound, addressed by COMPARE
Upper bound, addressed by, COMPARE FOR RANGE
Increasing value of accumulator contents
Indicator readings after COMPARE FOR RANGE
AL
AE
AH

Figure 16. Compare for Range Operation

PROGRAMMING EXAMPLES

1. Given a signed 10-digit decimal integer (byte size four) in location 1492. Store it in location 1500 if it is in the range 10^3 through 10^8, inclusive (Figure 17).

NAME	STATEMENT	NOTES
	L (D, 44, 4), 1492.0,0	
	KI (D, 20, 4), TEN3, 0	
	KR (D, 40, 4), TEN8, 0	
	BZAE, OUT	
	ST (D, 40, 4), 1500.0, 0	1
OUT	Continuation of program	
TEN3	DDI (D, 20, 4), 1000	
TEN8	DD (D, 40, 4), 100000001	

Note: 1. The stored quantity cannot exceed 9 digits, so a field length of 40 is used.

Figure 17, Compare Example

2. Given a signed 31-digit decimal integer stored in storage such that the most significant portion is in location 1935 and the least significant portion and sign are in location 1936. Store this number in locations 2000 and 2001 if it is in the range 10^{15} through 20^{25}, inclusive (Figure 18). Here, because of the double-length number, COMPARE FOR RANGE cannot be used. A combination of BRANCH ON INDICATOR and COMPARE IF EQUAL is used instead.

Compare Field for Range (KFR)

If the accumulator-high indicator is off when this operation is interpreted, the comparison is not performed and the comparison result indicators are not changed. Otherwise, the operation is identical to COMPARE FIELD with the exception of the setting of the comparison result indicators. If the accumulator operand is equal to or higher than the storage operand, the AH indicator is turned on. If the accumulator operand is less than the storage operand, the AE indicator is turned on.

NAME	STATEMENT	NOTES
	L (D, 64, 4), 1936.0, 0	1
	BRLZ, OUT	2
	C0011 (B, 64, 8), 1935.0, 60	3
	BZRGZ, OUT	4
	K (DU, 64, 4), TEN10, 60	5
	BAH, OUT	7
	KEI (DU, 60, 4), 0, 0	6
	BAH, OUT	7
	ST (DU, 64, 4), 2000.0, 60	
	ST (D, 64, 4), 2001.0, 0	
OUT	Continuation of program	
TEN10	DD (DU, 64, 4), 10.0 E10	

Notes:
1. Loads low-order part of number.
2. Tests for negative number.
3. Loads high-order part of number.
4. If number is less than 10^{15}, then high-order part is zero.
5. Compare with high-order portion of 10^{25}.
6. Compare with low-order portion of 10^{25}.
7. Branch if number is above range.

Figure 18. Compare Example

MODIFIERS

Unsigned; Negative Sign; Radix. These modifiers operate as in ADD.

INDICATORS

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Comparison Result (AL, AE, AH). These indicators are set as in COMPARE FOR RANGE.

PROGRAMMING NOTE

COMPARE FIELD FOR RANGE can be used with COMPARE FIELD to determine whether a quantity falls within a given range in the same manner that COMPARE FOR RANGE is used with COMPARE.

Load Transit and Set (LTRS)

The transit register (location 15), the left zeros counter, and the all-ones counter are cleared. A four-bit sign byte, in which the leftmost bit is set to the sign of the addressed storage operand as modified by the negative sign modifier, is placed in the four low-order bits of the transit register. The remaining three bits of this sign byte are zero. The numeric part of the addressed operand is positioned immediately to the left of this sign byte in the transit register. The effective offset specified by the instruction is placed in the all-ones counter (bits 44-50 of location 7). The two high-order bits of the left zeros counter (bits 17-18 of location 7) are set to ones.

At the end of the operation, either the binary-transit or decimal-transit indicator is turned on, depending on the value of the radix modifier.

MODIFIERS

Unsigned; Negative Sign. These modifiers operate as in ADD.

Radix. This modifier operates as in ADD. In addition, it determines whether the binary transit (BTR) or decimal transit (DTR) indicator is turned on.

INDICATORS

Partial Field (PF). Since the low-order four bits of the transit register are used as a sign byte, only 60 numeric bits can be loaded into it. This indicator is turned on if the numeric part of the field loaded contains more than 60 significant bits. If this occurs, the transit register is loaded up to its left end. Higher-order bits are lost.

Binary Transit (BTR); Decimal Transit (DTR). One of these two indicators is turned on at the end of the operation, depending upon the value of the radix modifier bit.

Data Flags (TF, UF, VF); To-Memory Operation (MOP). These indicators are set as in ADD.

Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set according to the contents of the transit register when the operation is complete.

PROGRAMMING NOTES

Since the two indicators, binary transit and decimal transit, have lower priority than any other indicators capable of causing an interruption, the necessary steps to correct any errors detected during LOAD TRANSIT AND SET operation will be taken before entering the sequence associated with the BTR or DTR indicators.

The primary purpose of this operation is to serve as an entry to an interpretive routine, for example, to a routine for an integer arithmetic square root operation. The effective offset, which has been placed in the all-ones counter, is available to the interpretive routine in the correct bit address to be directly transmitted to an index register and used to produce the desired effective offset.

The offset field of the instruction may also be used to distinguish between one of 128 pseudo-operations. For this purpose it is at the correct bit address to be used directly in indexing to a table of half-word BRANCH instructions.

The two high-order bits of the left zeros count may be used after an interruption caused by the DTR indicator to distinguish this operation from decimal MULTIPLY, DIVIDE, or MULTIPLY AND ADD, which may also turn on this indicator. These two bits are set as follows by the four operations:

MULTIPLY	00
DIVIDE	01
MULTIPLY AND ADD	10
LOAD TRANSIT AND SET	11

These bits are at the correct bit address to be used directly in indexing to a table of half-word BRANCH instructions.

Note that, if the interruption mechanism is disabled or the BTR or DTR indicators are masked off, the contents of the left-zeros and all-ones counters may have been altered since these indicators were turned on.

Multiply (*)

If binary is specified, the product of the addressed operand and the accumulator field specified by the offset is placed into the cleared accumulator at offset 20. Neither operand may exceed 48 significant numeric bits. If decimal is specified, this operation has the same action as LOAD TRANSIT AND SET, except that the two high-order bits of the left zeros counter are set to zeros.

MODIFIERS

Unsigned; Negative Sign. These modifiers operate as in ADD.

Radix. This modifier operates as in ADD. It also determines whether a binary multiplication is performed or whether instead the action is like that of a decimal LOAD TRANSIT AND SET.

INDICATORS

Partial Field (PF). This indicator is turned on in a binary operation if either operand exceeds 48 significant bits in length. In such a case, only the 48 low-order bits of the oversized operand are used in the operation. If the accumulator operand was oversized, the higher-order bits are lost. In a decimal operation, this indicator is set as in LOAD TRANSIT AND SET.

Decimal Transit (DTR). This indicator is turned on if a decimal operation is specified.

Data Flags (TF, UF, VF); To-Memory Operation (MOP). These indicators are set as in ADD.

Arithmetic Result (RLZ, RZ, RGZ, RN). In a binary operation, these indicators are set as in ADD. In a decimal operation, they are set as in LOAD TRANSIT AND SET.

PROGRAMMING NOTE

A decimal MULTIPLY instruction can be used to produce an automatic entry to a subroutine, where the desired decimal multiplication may be performed using the conversion operations and binary multiplication. Thus, if the proper subroutine is provided to handle interruptions by the decimal-transit indicator, the decimal MULTIPLY instruction may be used in a program as though this action were actually implemented in the machine. The subroutine has the two factors available in the accumulator and the transit register and the effective offset available in the all-ones counter.

Load Factor (LFT)

The factor register (FT) is cleared. A four-bit sign byte, in which the leftmost bit is the sign of the addressed storage operand as modified by the negative sign modifier, is placed in the four low-order bits of FT. The remaining three bits of this sign byte are zero. The numeric part of the addressed operand is positioned to the left of this sign byte in FT.

MODIFIERS

Unsigned; Negative Sign; Radix. These modifiers operate as in ADD.

INDICATORS

Partial Field (PF). Since the four low-order bits of FT are used as a sign byte, only 60 numeric bits can be loaded into FT. This indicator is turned on if the numeric part of the field loaded contains more than 60 significant bits. If this occurs, FT is loaded up to its left end. Higher-order bits are lost.

Data Flags (TF, UF, VF); To-Memory Operation (MOP). These indicators are set as in ADD.

Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set according to the contents of the factor register when the operation is complete.

PROGRAMMING NOTES

This operation is necessary to load the multiplicand of a subsequent MULTIPLY AND ADD. This instruction is not used to load either factor of a MULTIPLY.

The factor register is addressable and, therefore, can be addressed as the storage operand of any operation. FT is a 64-bit register with word address 14.

LOAD FACTOR is the only operation that changes the contents of FT as an integral part of its performance. FT is not changed by any other operation unless it is addressed explicitly.

The accumulator contents are not affected by LOAD FACTOR. The offset has no meaning in this operation and is ignored. The lost-carry indicator cannot be turned on by this operation.

Multiply and Add (*+)

If a binary operation is specified, the multiplicand field in the factor register (FT) is multiplied by the addressed storage operand. Neither multiplicand nor multiplier may exceed 48 significant numeric bits. The product is algebraically added to the accumulator contents as specified by the offset. If a decimal operation is specified, this operation has the same action as LOAD TRANSIT AND SET, except that the two high-order bits of the left zeros counter are set to 10.

MODIFIERS

Unsigned; Negative Sign. These modifiers operate as in ADD.

Radix. This modifier operates as in ADD. It also determines whether a binary multiplication and addition are performed or whether instead the action is like that of a decimal LOAD TRANSIT AND SET.

INDICATORS

Lost Carry (LC). In a binary operation, this indicator is set as in ADD. Lost carry cannot be turned on if decimal is specified.

Partial Field (PF). This indicator is turned on in a binary operation if either the multiplicand or the multiplier exceeds 48 significant bits in length. In such a case, only the 48 low-order bits of the oversized operands are used in the operation. Partial field is also turned on in a binary operation if the offset is such that significant bits of the product attempt to combine with accumulator positions to the left of the left end of the accumulator. Such high-order bits of the product are lost. In a decimal operation, partial field is set as in LOAD TRANSIT AND SET.

Decimal Transit (DTR). This indicator is turned on if decimal is specified.

Data Flags (TF, UF, VF); To-Memory Operation (MOP). These indicators are set as in ADD.

Arithmetic Result (RLZ, RZ, RGZ, RN). In a binary operation the above indicators are set as in ADD. In a decimal operation, they are set as in LOAD TRANSIT AND SET.

PROGRAMMING NOTES

At the completion of the operation, FT still contains the multiplicand and its sign. It is unchanged by the operation. The product that was formed during the operation is not available anywhere. To retain both the product and the cumulative sum, the instructions MULTIPLY and ADD TO MEMORY can be used.

The LOAD FACTOR operation should normally precede MULTIPLY AND ADD. However, the loading of FT need not be done in the operation immediately preceding the MULTIPLY AND ADD operation.

The offset specified has no effect upon the multiplication. The offset affects the addition of the product to the accumulator contents in the same manner that it affects ADD.

The arithmetic result indicators are set according to the entire contents of the accumulator when the operation is complete, and not according to the product.

Divide (/)

If a binary operation is specified, the entire accumulator contents to the left of the offset are divided by the addressed storage field. Accumulator positions to the right of the offset are ignored. At the completion of the operation, the quotient is placed in the cleared accumulator at the specified offset, and the remainder with sign is placed in the remainder register (RM). Both dividend and divisor are treated as integers, and only the integral part of the quotient is developed. The length of the dividend to the left of the offset cannot exceed 96 significant bits, and the divisor cannot exceed 48 significant numeric bits. The sign of the quotient replaces the sign bit in the accumulator sign byte register. Bits 60-63 of RM form a four-bit remainder sign byte with bit 60, the sign bit, set to the original accumulator sign. Bits 61-63 are set to zero. Bits 0-59 of RM are replaced by the integer remainder with the low-order remainder bit in bit position 59 of the remainder register.

If a decimal operation is specified, this operation has the same action as LOAD TRANSIT AND SET, except that the two high-order bits of the left zeros counter are set to 01.

Modifiers

Unsigned; Negative Sign. These modifiers operate as in ADD.

Radix. This modifier operates as in ADD. It also determines whether a binary division is performed or whether instead the action is like that of a decimal LOAD TRANSIT AND SET.

Indicators

Partial Field (PF). In a binary DIVIDE, this indicator is turned on if there are more than 96 significant bits in the accumulator to the left of the offset or if the divisor exceeds 48 significant bits. In such a case, only the low-order 96 or 48 bits are used in the operation and higher-order bits are ignored. Such high-order dividend bits in the accumulator are lost in the operation. In a decimal operation this indicator is set as in LOAD TRANSIT AND SET.

Zero Divisor (ZD). This indicator is turned on if the binary divisor is zero. The division is not attempted, and both the accumulator and the remainder register remain unchanged.

Decimal Transit (DTR). This indicator is turned on if decimal is specified.

Data Flags (TF, UF, VF); To-Memory Operation (MOP). These indicators are set as in ADD.

Arithmetic Result (RLZ, RZ, RGZ, RN). In a binary operation, these indicators are set as in ADD. In a decimal operation, they are set as in LOAD TRANSIT AND SET.

Programming Notes

The divide operation takes place and produces the correct result in every situation in which the operands do not exceed the maximum allowable length with the exception of the case of a zero divisor. The programmer need not be concerned with the relative magnitudes of the divisor and the dividend. The only scaling necessary is that always required to assure sufficient zeros on the right end of the dividend so that the desired length of quotient can be developed. This is readily accomplished by loading the dividend from storage with the proper offset. To divide two integers and obtain only the integral part of the quotient, no scaling is necessary. A meaningful quotient will be obtained in every case without scaling. The integral quotient is found in the accumulator at the completion of the operation with its low-order position at the specified offset.

The rightmost digit of the quotient is always found at the offset specified in the instruction. The bits to the right of the offset are zero.

The arithmetic result indicators describe the quotient, and not the remainder.

Binary point location is exceptionally simple in this division operation. The rules used in ordinary long division as performed by hand suffice. Thus, the ratio is expressed in a form in which the divisor appears as an integer. Effectively, the number of places to the right of the binary point in the quotient is equal to the number of places in the dividend minus the number of places in the divisor. For example, in decimal:

$$32.1\ \sqrt{6352.401} \quad = \quad 321_{\curvearrowright}.\ \sqrt{63524_{\curvearrowright}.01}$$

The following examples illustrate the ease of determining the divisor offset required. By divisor offset is meant the offset specified in the DIVIDE instruction.

1. Divide 1010.110 by 100.1 and obtain two binary places in the result. Assume that the dividend is already loaded into the accumulator at zero offset. Thus, the divisor may be considered as an integer:

$$100.1\ \sqrt{1010.110} \quad = \quad 1001_{\curvearrowright}.\ \sqrt{10101_{\curvearrowright}.10}$$

Then:

Divisor offset required = number of binary places in the dividend minus the number required in the quotient.

= 2 — 2

= 0 bits

After the division, the quotient of 10.01 will be found at zero offset in the accumulator, and the remainder of 0.101 in the low-order positions of RM.

2. Divide 11.0110110 by 10.011 and obtain two binary places in the result. Assume that the dividend is already loaded into the accumulator at zero offset.

$$10.011\sqrt{11.0110110} = 10011.\sqrt{11011.0110}$$

Divisor offset required = number of binary places in the dividend minus the number required in the quotient.

= 4 — 2

= 2 bits

Thus, an offset of two should be specified in DIVIDE. No scaling of the dividend is needed prior to DIVIDE because the divisor offset is non-negative. A quotient of 1.01 at offset two in the accumulator and a remainder of 0.01110 will be obtained. Note that the remainder has the same number of binary places as the portion of the original contents of the accumulator to the left of the offset. The two low-order positions of the dividend to the right of the offset are not used in the operation and are lost.

3. Divide 11.1011 by 101101.1 and obtain six binary places in the result.

$$101101.1\sqrt{11.1011} = 1011011.\sqrt{111.011}$$

Divisor offset required = number of binary places in the dividend minus the number required in the quotient

= 3 — 6

= —3 bits

Since a negative offset cannot be specified, it is necessary to scale the dividend prior to the DIVIDE. In this example, the dividend would be loaded into the accumulator with an offset of three bits, and zero offset would be specified in the DIVIDE. A quotient of 0.000101 at offset zero in the accumulator and a remainder of 0.0010001 would be obtained.

4. Divide 11001110.1 by 10.11 developing the quotient only as far as the "four" digit (effective number of binary places required is — 2).

$$10.11\sqrt{11001110.1} = 1011.\sqrt{110011101(0).}$$

Divisor offset required = number of binary places in the dividend minus the number required in the quotient

= — 1 — (— 2)

= 1 bit

Thus, an offset of one should be specified in the DIVIDE. No scaling of the dividend prior to the DIVIDE is required. A quotient of 10010 (with the binary point two places to the right) is obtained positioned at an offset of one in the accumulator, and a remainder of 1000. in the low-order positions of the remainder register.

In general, divisor offset minus dividend offset is equal to the number of binary places in the dividend, minus the number in the divisor, minus the number in the quotient:

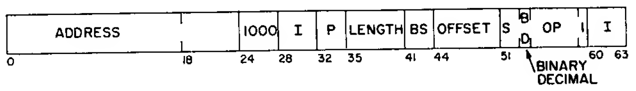
$$of_{dr} - of_{dd} = b_{dd} - b_{dr} - b_q$$

Radix Conversion Operations

Any one of four radix conversion operations may convert an integer either from binary to decimal or from decimal to binary. Two of these operations, LOAD CONVERTED and LOAD TRANSIT CONVERTED, obtain a field from storage, convert it, and place the result in the accumulator or the transit register, respectively. The other two operations, CONVERT and CONVERT DOUBLE, take a field from the accumulator, convert it, and return it to the accumulator. These latter two operations differ from each other in the length and position of the binary field involved.

Format

The radix conversion operations use the same full-word instruction format as the integer arithmetic operations. In this case, bit 53, the radix modifier, describes the format of the original field and also determines the type of conversion, binary to decimal or decimal to binary.



Field Definition

The storage operand in LOAD CONVERTED and LOAD TRANSIT CONVERTED is defined by the effective address, field length, and byte size in the same manner as in integer arithmetic. Immediate addressing is also handled in the same way in these radix conversion operations as in the arithmetic operations.

In the accumulator conversion operations, CONVERT and CONVERT DOUBLE, no storage operand is required, and the word address, bit address, length, and byte size fields of the instruction are not normally used. If the conversion is from binary to decimal, the binary field is taken from a fixed position in the accumulator, and the decimal result is put back at a position specified by the offset. If the conversion is from decimal to binary, the decimal field is taken from the position in the accumulator specified by the offset, and the binary result is put back at a fixed position.

Sign Control

The radix conversion operations include two sign modifier bits which operate in the same manner as in the integer arithmetic operations. In all cases these modifiers apply to the original field before conversion and specify how its sign is to be modified during the conversion. All possible values may be obtained for the sign of the result.

Radix

Bit 53 of the instruction, the radix modifier, specifies the radix of the operand to be converted, and hence also whether binary-to-decimal or decimal-to-binary conversion is to be performed. If this bit is zero, the original field is binary; if this bit is one, the original field is decimal. This modifier also specifies the interpretation of the byte size field in the same manner as in integer arithmetic operations. Byte size four is assumed in the accumulator conversion operation.

Indicators

A number of indicators in the indicator register are set as a result of radix conversion operations. These include the partial-field indicator and the data-flag and arithmetic-result indicators. The partial-field indicator is permanent. If it is turned on by any operation it will remain on until it either causes an interruption or is explicitly set to zero. The data-flag and arithmetic-result indicators are temporary. Those affected by a given operation are set at the end of the operation to reflect the result of that operation. They then remain as set until the execution of another operation that affects them. The indicators that are affected by the radix conversion operations are described below.

Partial Field (PF). This indicator is turned on if a radix conversion operation does not use all of the operand data provided. This may occur if significant result bits extend beyond the left end of the accumulator or if either the operand or result contains more significant bits than can be handled by the processing unit. The conditions that will cause a partial field indication are listed in the description of each operation.

Data Flags (TF, UF, VF). In a radix conversion operation using a storage operand, these indicators are set according to the data flag bits, if any, of the storage operand. Any operation that normally affects these indicators will turn them off if the unsigned modifier is designated in the operation. These indicators are not affected by CONVERT or CONVERT DOUBLE.

To-Memory Operation (MOP). This indicator is turned off by all radix conversion operations.

Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set according to the final result of the radix conversion operation.

Operations

The four radix conversion operations are shown in the table below, which indicates the source of the operand and the register in which the result is placed.

		OPERAND TAKEN FROM	RESULT PLACED IN
LCV	Load Converted	Storage	Accumulator
LTRCV	Load Transit Converted	Storage	Transit
CV	Convert	Accumulator	Accumulator
DCV	Convert Double	Accumulator	Accumulator

Load Converted (LCV)

The addressed storage field is converted as an integer from decimal to binary or from binary to decimal depending on the setting of the radix modifier. The accumulator is cleared and the converted field is placed into it at the specified offset. The four low-order bits of the sign byte register are cleared. The sign bit in the sign byte register is set according to the sign of the data as affected by the sign modifiers.

MODIFIERS

Unsigned; Negative Sign. These modifiers operate as in ADD.

Radix. If decimal is specified, the storage operand is converted from decimal to binary after conversion of the decimal field to byte size four. If binary is specified, the storage operand is converted from binary to decimal, with the decimal result in byte size four.

INDICATORS

Partial Field (PF). This indicator is actuated when significant bits of the converted field extend beyond the left end of the accumulator. In such a case, the accumulator is filled up to its left end, and higher-order bits are lost. In a decimal-to-binary conversion, this indicator is also actuated if the storage field exceeds 15 decimal digits (as could be specified using a byte size of less than four). Meaningless results are then obtained.

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set as in ADD.

Load Transit Converted (LTRCV)

The addressed storage field is converted as an integer from decimal to binary or from binary to decimal depending on the setting of the radix modifier. The transit register, storage location 15, is cleared. The four low-order bits of the transit register form a sign byte in which the leftmost bit is the sign of the addressed storage operand as modified by the negative sign modifier. The remaining three bits of the sign byte are zero. The converted field is placed in the transit register positioned to the left of the four-bit sign byte. The accumulator is not altered by this operation, and the offset field of the instruction is ignored.

MODIFIERS

Unsigned; Negative Sign; Radix. These modifiers operate as in LOAD CONVERTED.

INDICATORS

Partial Field (PF). This indicator is actuated if the result in a binary-to-decimal conversion exceeds 15 significant decimal digits or if the result of a decimal to binary conversion exceeds 48 significant bits. In the former case the result is placed in the transit register up to its left end. In the latter case, only the 48 low-order bits of the result are placed in the transit register. This indicator is also actuated if the storage field specified in a decimal-to-binary conversion exceeds 15 decimal digits. Meaningless results are then obtained.

Data Flags (TF, UF, VF); To-Memory Operation (MOP). These indicators are set as in ADD.

Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set according to the final result in the transit register.

Convert (CV)

A field from the accumulator is converted as an integer from binary to decimal or from decimal to binary depending on the setting of the radix modifier. If the original field is binary, it is obtained at an implied offset of 68 and its length is 48 numeric bits. The decimal result is loaded into the cleared accumulator at the specified offset. If the original field is decimal, it is obtained at the specified offset. The binary result, which can be a maximum of 48 significant bits, is loaded into the cleared accumulator at offset 68. The sign modifiers operate on the sign bit in the sign byte register; all other positions of the sign byte register remain unchanged.

MODIFIERS

Unsigned. If the unsigned modifier is zero, the accumulator sign is taken as-is before the negative sign modifier is applied. If the modifier is one, the accumulator sign is taken as positive before the negative sign modifier is applied.

Negative Sign. If the negative sign modifier is zero, the accumulator sign is taken as-is after modification by the unsigned modifier. If the negative sign modifier is one, the accumulator sign after modification by the unsigned modifier is inverted.

Radix. If binary is specified, the 48-bit accumulator field at offset 68 is converted from binary to decimal. Higher-order bits (positions 0-11 of the accumulator) are ignored and lost. No indication of this loss is retained. The decimal result in byte size four is placed into the accumulator at the specified offset. If decimal is specified, the accumulator field at the specified offset is converted from decimal to binary. The binary result is placed into the accumulator at offset 68. The original decimal field is assumed to be byte size four.

INDICATORS

Partial Field (PF). This indicator is actuated in a decimal-to-binary conversion if the binary result exceeds 48 significant bits. In such a case, the low-order 48 bits of the result are placed in the accumulator. Any higher-order bits are lost. The indicator is actuated in a binary-to-decimal conversion if significant bits of the decimal result extend beyond the left end of the accumulator when this result is placed into the accumulator at the specified offset.

To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set as in ADD.

Convert Double (DCV)

A field from the accumulator is converted as an integer from binary to decimal or from decimal to binary depending on the setting of the radix modifier. If the original field is binary, it is obtained at an implied offset of 20 and its length is 96 numeric bits. However, no more than 80 significant bits can be converted without causing a partial field condition. The decimal result, which can be a maximum of 24 significant digits, is loaded into the cleared accumulator at the specified offset. If the original field is decimal, it is obtained at the specified offset. The binary result, which can be a maximum of 96 significant bits, is loaded into the cleared accumulator at offset 20. The sign modifiers operate on the sign bit in the sign byte register; all other positions of the sign byte register remain unchanged.

MODIFIERS

Unsigned; Negative Sign. These modifiers operate as in CONVERT.

Radix. If binary is specified, the 96-bit accumulator field at offset 20 is converted from binary to decimal. Higher-order bits (positions 0-11 of the accumulator) are ignored and lost. No indication of this loss is retained. The decimal result in byte size four is placed into the accumulator at the specified offset. If decimal is specified, the accumulator field at the specified offset is converted from decimal to binary. The binary result is placed into the cleared accumulator at offset 20. The original decimal field is assumed to be byte size four.

INDICATORS

Partial Field (PF). This indicator is actuated if the converted field exceeds 96 significant bits in either type of conversion. The 96 low-order bits of the result are obtained, and higher-order bits are lost. The indicator is also actuated in a binary-to-decimal conversion if significant bits of the decimal result extend beyond the left end of the accumulator when this field is loaded into the accumulator at the specified offset. Since a maximum of 24 decimal digits can be produced, no more than 80 significant bits can be converted without causing a partial field condition.

To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set as in ADD.

PROGRAMMING NOTES

The effective address of the first half-word, the byte size field, and the length field are not normally used by either the CONVERT or the CONVERT DOUBLE operations and are ignored. If progressive indexing is specified, the specified index register will be modified as specified by the progressing indexing code in the normal manner.

The definition of the sign modifiers permits complete control over the accumulator sign. It may be left as-is, inverted, made positive or made negative.

The fixed offsets used in the accumulator conversion operations are designed to facilitate conversion between binary floating point and decimal formats. Thus, the 48-bit binary field at offset 68 used in CONVERT corresponds to the fraction field of a single-precision floating point number; and the 96-bit binary field at offset 20 used in CONVERT DOUBLE corresponds to the fraction field of a double-precision floating point number. This 96-bit field also corresponds to the result field of a binary integer MULTIPLY instruction.

PROGRAMMING EXAMPLE

The problem of multiplying two decimal integers illustrates the use of the radix conversion operations (Figure 19). Given two four-digit signed decimal integers (byte size six) in locations 1900 and 1900.30, form the eight-digit product and store it in location 2000.

NAME	STATEMENT	NOTES
	LCV (D, 30, 6), 1900.0, 8	
	LTRCV (D, 30, 6), 1900.30, 0	1
	* (B, 64, 4), $TR, 8	2
	DCV (B, 0, 8), 0, 0	3
	ST (D, 54, 6), 2000.0,0	

Notes: 1. Converts multiplier and places it in transit register.
2. The offset of the LCV and the * instruction should be the same. A byte size of four is used in * because of the four-bit sign byte in the transit register.
3. DCV is used instead of CV because of the offset of 20 produced by the * instruction.

Figure 19. Radix Conversion

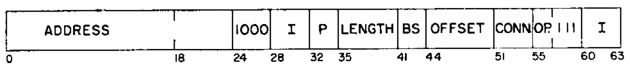
Connective Operations

Each of three connective operations can specify any one of the sixteen binary connectives. One operand of these operations is in the accumulator. The other is in storage or in the instruction itself. The accumulator operand extends from a low-order position specified by the offset and consists of the same number of bytes as are in the storage operand. The storage operand is defined by an address and field length as in integer arithmetic. In each of the connective operations, the two operands are combined according to the logical connective specified in the instruction.

The result obtained is used to develop certain tests. In CONNECT, the result replaces the accumulator operand; in CONNECT TO MEMORY, the result replaces the storage operand; in CONNECT FOR TEST, the result is discarded after developing the tests.

Format

The connective operations use basically the same full-word instruction format as the integer arithmetic operations. The word and bit address, the two I fields, the P field, the field length, byte size, and offset occupy the same bit positions in the instruction format and have the same function in the connective operations as they do in the integer arithmetic operations.



The operation code in bits 55-59 designates the operation to be performed. The connective that is to be applied is specified in bits 51-54.

Field Definition

The operand specified explicitly by a connective instruction is defined by the effective address in the same manner as in integer arithmetic. Immediate addressing is also handled in the same way in the connectives as in the arithmetic operations. The same is true of field extraction from and field insertion to storage.

The implied second operand of the connective operations is a field from the accumulator. The right end of this operand is defined by the offset specified in the instruction. In contrast to the integer arithmetic operations, the accumulator operand has a definite length; it is composed of as many bytes as there are in the addressed storage field. In CONNECT, only those accumulator bytes which are combined with the storage operand can be affected. The remainder of the accumulator contents remains unchanged.

In CONNECT TO MEMORY, the number of accumulator bytes between the specified offset and the left end of the accumulator can be less than the number of bytes in the storage operand. In this event, the accumulator operand is lengthened for the operation by the addition of high-order zeros.

Byte Size

The connective operations process data in eight-bit bytes. In these operations, the byte size specified in the instruction designates the number of bits used to represent each character or symbol in the storage field. If the byte size of the storage operand is less than eight, each byte is converted into an eight-bit byte before processing by the addition of high-order zeros. The accumulator operand is assumed to have a byte size of eight. Any offset may be specified and the field is processed in eight-bit bytes starting with the offset. Each byte of the storage field, after conversion to eight-bit form if necessary, is combined with an eight-bit byte from the accumulator according to the specified connective. Consequently, the result bytes always have a byte size of eight. In CONNECT, these result bytes replace the accumulator field which participated in the operation. In CONNECT FOR TEST, the result field is discarded after developing certain tests. In CONNECT TO MEMORY, the result bytes replace the storage operand in accordance with the byte size of that operand. The storage field may have any byte size from one to eight. Since each result byte replaces the corresponding byte in the storage field, the result byte is truncated before being placed in storage if the byte size of the storage field is less than eight. Each byte in the storage field is replaced by the number of bits specified by the byte size. These bits are the low-order bits of each result byte; the remaining high-order bits of each result byte are dropped. No indication is given if the bits which are dropped are significant.

Since the field lengths of the storage operand need not be a multiple of the byte size, the last byte need not be of full size. When this is the case, only those bits of the high-order accumulator and storage bytes that are included in the field length take part in the processing. No high-order zeros are added to the short byte from storage. However, if the field length is a multiple of the byte size, and if the byte size is less than eight, high-order zeros are added to the high-order byte as well as to all other bytes so that all bytes of the field are treated in the same manner.

To combine a group of bits in storage with the same number of contiguous bits in the accumulator, a byte size of eight should be specified. This avoids the addition of high-order zeros to each byte of the storage operand.

Counts and Indicators

Two counts describing the result field are available after a connective operation. The seven-bit all-ones counter, bits 44-50 of location 7, contains a count of all ones in the result field. The seven-bit left-zeros counter, bits 17-23 of location 7, contains a count of the number of zeros in the result field that are to the left of the most significant one-bit. These counts are positioned so that they can readily be used for indexing. The position of the left-zeros counter in the first half word of location 7 is such that it corresponds to that part of an index value field which indexes the bit address of an instruction. The position of the all-ones counter in the second half-word of location 7 is such that it facilitates indexing of half-word addresses. Both counters operate modulo 128. Each count is on the final result field. In CONNECT and CONNECT FOR TEST, the counts are made on the result field with byte size eight. In CONNECT TO MEMORY, the counts are made on the result field after it is converted to the byte size of the storage operand. The high-order bits that are dropped from each result byte if the byte size is less than eight are ignored in developing the counts. Positions in the accumulator that do not take part in the operation have no effect on the counts. The left-zeros and all-ones counts are destroyed by all LOAD TRANSIT AND SET operations and by decimal MULTIPLY, MULTIPLY AND ADD, and DIVIDE operations. The left-zeros count is also destroyed by all floating-point division operations. These counts are not changed as an inherent part of any other non-connective operations. The counts are set at the end of the connective operations, so their previous values may be used as operands in the operation.

The indicators that are affected by the connective operations are described below.

Partial Field (PF). This indicator is turned on if the storage field attempts to combine with bits beyond the left end of the accumulator. If a partial-field condition occurs, the storage operand is combined with the accumulator operand up to the left end of the accumulator; higher-order bits in the storage operand are dropped. In such a case, the counts are developed and the result indicators are set according to that portion of the result actually developed. The partial-field indicator is not affected by CONNECT TO MEMORY.

To-Memory Operation (MOP). This indicator is turned on by CONNECT TO MEMORY and turned off by CONNECT and CONNECT FOR TEST.

Result Zero (RZ). This indicator is turned on if the final result is all binary zeros. This implies that the all-ones count is also zero and that the left-zeros count is equal to the number of bits in the final result field. In CONNECT and CONNECT FOR TEST, this number is equal to the number of accumulator bits which take part in the operation. In CONNECT TO MEMORY, this number is equal to the number of bits in the storage operand, since this is the length of the final result field. Only those bits which replace the storage operand affect the two counts and the result indicators. The high-order bits that are dropped from each result byte if the byte size is less than eight are ignored.

Result Greater than Zero (RGZ). This indicator is turned on if the result field contains any one-bit.

Result Less than Zero (RLZ); Result Negative (RN). These indicators are turned off by the connective operations.

Connectives

The sixteen connectives that can be specified are listed in Figure 20. The storage (memory) operand is denoted by *m*; the accumulator operand by *a*.

	Result Bit for Operand Bit Combinations				Symbolic Representation of Logical Function
	m a 0 0	m a 0 1	m a 1 0	m a 1 1	
0	0	0	0	0	0
1	0	0	0	1	$m \cdot a$
2	0	0	1	0	$m \cdot \bar{a}$
3	0	0	1	1	m
4	0	1	0	0	$\bar{m} \cdot a$
5	0	1	0	1	a
6	0	1	1	0	$m \veebar a$
7	0	1	1	1	$m \vee a$
8	1	0	0	0	$\bar{m} \cdot \bar{a}$
9	1	0	0	1	$m \equiv a$
10	1	0	1	0	$\bar{a}$
11	1	0	1	1	$m \vee \bar{a}$
12	1	1	0	0	$\bar{m}$
13	1	1	0	1	$\bar{m} \vee a$
14	1	1	1	0	$\bar{m} \vee \bar{a}$
15	1	1	1	1	1

Figure 20. Connectives

The four-bit code used for the logical connectives is composed of the result bits obtained for each of the possible combinations of *m* and *a*.

In this code, the first bit represents the result when *m* and *a* are both zero; the second bit represents the result when *m* is zero and *a* is one; the third bit represents the result when *m* is one and *a* is zero; and the fourth bit represents the result when *m* and *a* are both one.

For example, if connective 0101 is specified, the result bit will be a one when and only when the *a* operand is a one-bit. If connective 1011 is specified, the result bit will be a one for all combinations of *m* and *a*, except *m* equal to zero and *a* equal to one.

Connect (C)

The addressed storage operand is combined with the accumulator field specified by the offset in accordance with the logical connective specified. The result field replaces the accumulator operand. The remainder of the accumulator remains unchanged.

Indicators

Partial Field (PF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set as described in "Counts and Indicators."

Connect to Memory (CM)

The accumulator field specified by the offset is combined with the addressed storage operand in accordance with the logical connective specified. The result replaces the storage operand. The accumulator contents are unchanged by the operation.

Indicators

To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set as described in "Counts and Indicators."

Connect for Test (CT)

The addressed storage operand is combined with the accumulator field specified by the offset in accordance with the logical connective specified, as in CONNECT. The result is discarded after the left-zeros and all-ones counts are developed and the indicators set. Both the accumulator contents and the storage operand remain unchanged.

Indicators

Partial Field (PF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set as described in "Counts and Indicators."

Programming Notes

The CONNECT operation has many uses other than that of evaluating logical conditions. The connectives 0000 and 1111 can be used to set the specified accumulator or storage bits to zero or one. A section of the accumulator can be replaced by connective 0011, while connective 0101 has no effect except to test the accumulator field and develop the counts. Connective 1010 inverts the accumulator operand.

Testing of storage fields for all zeros is readily performed by CONNECT FOR TEST 0011, while testing for all ones requires connective 1100. In each case, the result-zero indicator will be set to one if the test is satisfied.

To test certain combinations of bits in the accumulator for zero or one, a mask must be furnished in an immediate address or in storage. If the mask uses 1 for bits to be tested and 0 for bits to be ignored, the operation for testing the masked bits for all zeros is CONNECT FOR TEST 0001. If a test for ones is desired, the connective 0010 should be used. In either case, satisfaction of the test is indicated by the result-zero indicator. If the field to be tested is in storage, the mask must be in the accumulator and connectives 0001 and 0100, respectively, must be used.

In to-accumulator integer arithmetic operations, the result indicators are set according to the entire contents of the accumulator when the operation is complete. This is not the case in the connective operations. In CONNECT and CONNECT FOR TEST, these indicators are not affected by accumulator bits which do not participate in the operation. For example, if a zero result field is obtained when the storage operand is combined with the accumulator operand, the result-zero indicator is turned on even if there are non-zero bits in the accumulator to the right or left of the accumulator field that was combined with the storage operand.

It is possible for all 128 bits of the accumulator to participate in a single connective operation if the specified offset is zero and the byte size and field length are such that the storage operand equals or exceeds 128 bits when converted to byte size eight. In this event, a zero all-ones count can indicate either no ones or 128 ones in the result of a CONNECT or CONNECT FOR TEST, because the counter operates modulo 128. The two cases may be distinguished by interrogation of the result-zero indicator. Similarly, a zero left-zeros count can indicate either no left zeros or 128 left zeros in the result. These two cases can also be distinguished by interrogation of the result-zero indicator.

The data-flag and lost-carry indicators are not affected by the connective operations.

Programming Example

Given a 128-bit quantity in the accumulator, shift the entire quantity left ten bits. Discard the high-order bits that are shifted out of the accumulator and shift zeros into the low-order positions. Figure 21 shows the use of the connective operations for purposes other than the evaluation of logical conditions.

NAME	STATEMENT	NOTES
	C0011 (BU, 54, 8) , 8.10, 74	1
	C0011 (BU, 64, 8) , 9.0, 10	
	C0000 (BU, 10, 8) , 0, 0	2

Notes: 1. Shifts the left half of the accumulator left ten bits. A field length of 54 is used to avoid a partial field condition from non-zero high-order bits that are discarded.
2. Replaces the ten low-order bits of the accumulator with zeros. The address of zero is a dummy.

Figure 21. Connective Operations

Floating-Point Arithmetic

A floating number consists of a signed exponent $\pm E$, and a signed fraction $\pm F$. The quantity expressed by this number is the product of the fraction and the number two raised to the power of the signed exponent, or $\pm F \times 2^{\pm E}$. The exponent is expressed as a binary integer and the fraction is expressed as a binary number having a binary point to the left of the high-order digit.

A major objective in the design of this system is maximum processing speed for floating point operations. Simplification of the floating point instruction set is achieved through full utilization of the uniform nature of floating point data. The necessary instruction information requires only a half-word which allows two floating-point instructions per storage word. This results in an increase of program storage efficiency and reduces the number of storage accesses. Uniform information such as bit address, field length, addressing mode, byte size, radix, and accumulator offset are implied by the operation and thus need not be specified as required in integer arithmetic. Operations on the floating point fractions are performed at very high speed in parallel arithmetic, and operands utilize fixed portions of the operand registers.

Floating point instructions contain sign modifiers similar to the integer arithmetic sign modifiers. They also contain a normalization modifier which specifies the choice between normalized and unnormalized operation.

A quantity can be represented by a single floating-point number with the greatest precision when that number is normalized. A normalized floating point number always has a one in the high-order fraction bit position. If one or more high order fraction bits are zero, the number is not in normalized form. The process of normalization consists of shifting the fraction left until the high-order bit is one, and reducing the exponent by the amount of the shift.

When a single floating-point word cannot express a quantity with sufficient precision, multiple-precision operation is required. The floating-point instruction set contains a number of operations providing double-precision results. These operations have been designed to facilitate the programming of double or higher precision operation.

In floating point operation, the fractions of all arithmetic results contain a standard number of bits. Not all of these bits need be significant; furthermore, as a result of calculation, the number of significant bits may be reduced. When the number of significant bits becomes too small, the arithmetic procedure may be revised, or operations using greater precision may be used.

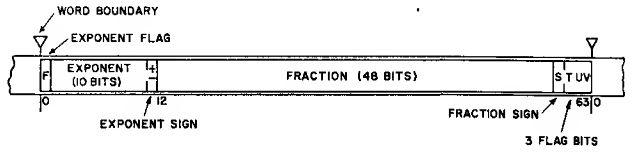
In order to simplify significance studies, a mode of floating-point operation is provided in which standard results are altered in a specific manner. This mode of operation is called "noisy mode." By computing a problem section in the standard mode and comparing the results with those obtained by computing the same problem section in the noisy mode, an estimate of the significance of the results can be obtained.

The system gives the best possible speeds for operands which are single-precision normalized numbers and results which are also normalized, since they represent the most frequent use of floating-point arithmetic. All other cases, however, are handled in a straightforward manner.

Floating-point numbers cover a range between the positive and negative values of the fraction having the maximum exponent. Since the exponent range is finite, a discontinuity exists between the positive and negative values of the fraction having minimum exponent. Included in this range is the number zero. A control or flag bit has been incorporated in the exponent field in order to provide straightforward control of data which exceed the exponent range or fall within the range of the discontinuity.

Data Format

Floating-point numbers are represented in storage in the following full word format.



The exponent field uses the first 12 bits of the data word, bit positions 0-11. Bit position 0 is called the exponent flag, EF. Bit positions 1-10 are called the exponent magnitude, EM. Bit position 11 is called the exponent sign, ES. The ES is set to 0 for positive values and set to 1 for negative values of the exponent. The EM is a binary integer in the range $0 \leqq \text{EM} \leqq 1023$. The EF is set to 0 for values of EM within the range. EF is set to 1 when EM exceeds this range.

The standard operational range for exponents is from +1023 through −1023. This range is called the normal, N, range. N range values have magnitudes for normalized floating point numbers ranging between 2^{+1024} and 2^{-1024}, or approximately 10^{+308} and 10^{-308}.

Through the use of the exponent flag, two additional ranges of numbers are provided. These are called the exponent flag positive, XFP, range and the exponent flag negative, XFN, range. The XFP range may be considered to have the properties of undefined numbers, sometimes symbolized by ∞. This range corresponds to all values with exponents greater than +1023. The XFN range may be considered to have the properties of the number zero. This range corresponds to all values with exponents less than −1023.

Algebraically, the use of these types of values in arithmetic operations produces undefined results. The requirements of machine processing require specific definitions for such cases. Therefore, the arbitrary definitions for these results have been selected such that result exponents will be propagated in the most noticeable direction.

The ranges discussed are classified as follows:

RANGE	EF	ES	DEFINITION
XFP	1	0	Exponent ≥ +1024
N	0	0 or 1	+1024 > Exponent > −1024
XFN	1	1	−1024 ≥ Exponent

The fraction field uses the next 49 bits of the data word, bit positions 12-60. Bit positions 12-59 are the fraction magnitude, F. Bit position 60 is the fraction sign, S. The binary point associated with the fraction is defined to be to the left of the high-order position, bit position 12. This means that a 1, in bit position 12, has the numerical value of one-half.

The data flag bits T, U, and V occupy bit positions 61, 62, and 63, respectively. These bits can be set by the programmer and, when set to 1, turn on the associated indicators. The four bits S, T, U, and V are treated as the sign byte for floating-point operations. In the accumulator sign byte register, they are stored in bit positions 4, 5, 6, and 7.

The floating point ranges specified above may be visualized through Figure 22, in which F represents the binary fraction and f represents the equivalent decimal fraction. The shaded areas represent the ranges of exceptional conditions that are handled by the system. These ranges may be considered as numerical buffers that assist in avoiding numerical difficulties encountered in going from machine processable values to non-machine processable values because of exponent overflow or underflow.

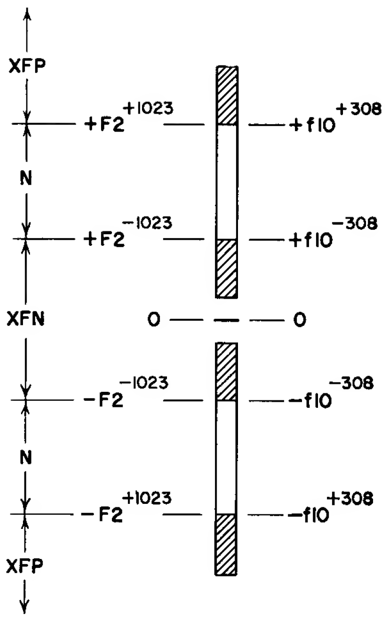
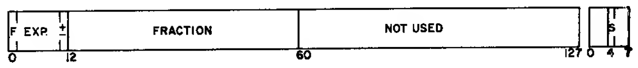


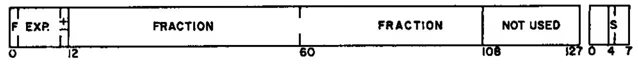
Figure 22. Floating Point Number Range

In the accumulator, floating-point numbers are represented either in single precision or in double precision. For single precision, the following format is used:



The exponent field is in bit positions 0-11. The fraction magnitude is in bit positions 12-59. The fraction sign bit does not appear in the accumulator register, but in bit position 4 of the accumulator sign byte register. Bit positions 5-7 of the accumulator sign byte register may contain the data flag bits associated with the information in the accumulator. Accumulator bit positions 60-63 are not used and remain unchanged during single-precision floating-point operations. The contents of the right half of the accumulator are neither used nor changed by a single-precision floating-point operation unless address 9 is used as the addressed operand. Bit positions 0-3 of the accumulator sign byte register are not used by any floating-point operation. They will not be changed by any floating-point operation unless address 10 is used as the addressed operand.

For double-precision operations, the following format is used:



The exponent field, high-order fraction magnitude and fraction sign occupy the same bit positions as indicated in single-precision operations. The low-order fraction magnitude of 48 bits is in bit positions 60-107. Bit positions 108-127 are not used or changed by any floating point operation, unless address 9 is used as the operand address.

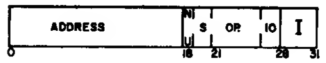
The accumulator is used as an implied operand for most operations. It is possible, however, to use the accumulator as the addressed operand as well as the implied operand. The left half of the accumulator has address 8; the right half of the accumulator has address 9; the accumulator sign byte register has address 10.

In operations other than floating point, the contents of the left half of the accumulator, bit positions 0-63, participate when address 8 is used as the effective operand address. However, in floating-point instructions when address 8 is used as the effective operand address, accumulator bit positions 0-59 and sign byte register bit positions 4-7 are used as the operand. The four sign byte bits replace the four low-order bits of the accumulator operand. Bits 60-63 of the accumulator do not serve as operand bits. For all floating-point single-precision operations, they remain unchanged. Bit positions 60-63 of the accumulator are changed in floating-point double-precision operations for which they are part of the implied operand and are changed as such. This rule applies to operations of both fetch and store type. Addresses 9 and 10, when used as effective operand addresses, are treated in floating-point as with other operations.

Floating-point remainders and cumulative multiplicands occurring in the remainder register (RM) address 13, and the factor register (FT) address 14 have the format of floating-point words in memory.

Instruction Format

Floating-point instructions are contained in the half-word format shown below. The address portion, bit positions 0-17, is the location of the full-word floating-point operand. Bit position 18 is used as the normalization modifier. Bit positions 19 and 20 are used as the sign modifiers. The operation code is specified in bit positions 21-27. Of these, bit positions 26 and 27 contain the fixed code $(10)_2$ to indicate that the instruction belongs to the floating-point class. The five remaining bits of the operation code, bit positions 21 through 25, allow 32 operations to be specified. Of these, 29 operations are used. The operand address can be modified by the value field of the index register specified in bit positions 28-31.



Single- and Double-Precision Arithmetic

Single-precision and double-precision operations differ in the number of bits used in the operands and developed in the final result fraction. The exponent field, in either case, has one EF bit, ten EM bits, and one ES bit.

In single-precision arithmetic, each operand has 48 fraction bits. During the execution of the operation, zeros are implied to the right of the specified fraction bits. Only bit positions 12-59 of the accumulator are used. The final result of a single precision arithmetic operation is also a 48-bit fraction. However, during the process of obtaining the final result, an *intermediate* result is developed in the arithmetic unit, which may have more than 48 fraction bits. During addition, a 96-bit sum fraction and a possible high-order carry, the fraction overflow bit, is obtained as an intermediate result. During multiplication, a 96-bit product fraction is obtained. During division, a 48-bit quotient fraction is obtained; no remainder is preserved. In the operations load, store and square root, the result fraction is 48 bits. These intermediate result fractions are adjusted to a final result fraction of 48 bits according to the rules associated with normalized or unnormalized operations.

In double-precision arithmetic operations, the accumulator contains a 96-bit fraction. The memory operand has 48 bits in the fraction. The result is placed in the accumulator and has 96 bits in the fraction. Intermediate results of 96 bits and a possible fraction overflow bit are developed except in the case of division.

In the operation DIVIDE DOUBLE, a 49-bit quotient fraction is developed. After the 48th quotient bit has been developed, the high-order 48 bits of the current remainder are stored in the remainder register, RM, with the proper exponent. When the remainder has been stored, it is compared with the divisor to obtain the 49th quotient fraction bit.

The result of an addition may be placed in storage in the single-precision add-to-memory operation. This operation is not possible in double-precisior operation, since the sum has more bits than the addressed operand.

All single-precision floating-point operations provide results which are truncated to 48 bits. In subsequent operations, zeros are implied to the right of the low-order bit. The magnitude of the computed result may be slightly reduced and the cumulative effect of truncation may result in a bias in the computation. By using rounded results rather than truncated results, such bias may be reduced. To obtain rounded single-precision results, a series of operations may be performed in double-precision, after which the instruction STORE ROUNDED is used to store the results. Rounding is achieved by adding 1 to the 49th bit of the result fraction, bit position 60, and truncating the result fraction to 48 bits.

Normalization

In floating point instructions, normalized or unnormalized operation is specified through use of the *normalization* modifier, instruction bit position 18. Normalized operation is specified when this bit is zero; unnormalized operation is specified when this bit is one.

When normalized operation is specified, the initial operands used in the operation need not be in normalized form, but the result of the operation is usually normalized. The exceptions to providing a normalized result are discussed with each operation. When normalization occurs, it may involve a left shift of the fraction to eliminate high-order zeros, or it may involve a right shift to insert a high-order one bit when the arithmetic operation produces a result fraction overflow bit. After the shift, the result fraction is truncated to 48 or 96 bits. When normalization of the result requires shifting, the exponent is changed by the amount of the shift. A right shift is added to the exponent; a left shift is subtracted from the exponent. An exception to the rules for normalization occurs when the 48-bit result fraction in single-precision operations or the 96-bit result fraction in double-precision is zero. For zero fractions the normalization is suppressed and no change, due to normalization, is made in the result exponent. This point is further discussed in "Zero Definition."

When unnormalized operation is specified, the arithmetic operation result fraction is truncated to 48 or 96 bits without a normalization cycle. High-order zeros in the fraction are not eliminated. If a fraction overflow bit is produced, the extra bit is lost and the lost carry indicator, LC, is turned on. The details of unnormalized operation are discussed with each operation.

Normalization, when specified by the instruction, usually applies to the result of the operation. This is called post-normalization. In performance of the arithmetic operations of addition and division, another form of normalization may be used. This is called prenormalization and is applied to one or more of the operands specified in the operation. During addition, the operand having the smaller exponent is shifted right with its exponent increased by the amount of the shift, until the exponents of the two operands agree in magnitude and sign. During division, both the divisor and dividend may be prenormalized. Prenormalization of the divisor occurs as a part of the arithmetic procedure and is independent of the normalization modifiers. Prenormalization of the dividend is required when a normalized quotient is to be obtained. A fully normalized dividend is developed when a normalized quotient is to be obtained. Dividend prenormalization is conditional on the setting of the normalization modifier. If an unnormalized quotient is to be obtained, the dividend fraction is shifted, at most, the same amount as the divisor.

A normalized result fraction, as developed by this system, has a magnitude in the range $1 > F \geqq 1/2$ or zero. An unnormalized result fraction has a magnitude in the range $1 > F \geqq 0$.

The unnormalized mode of operation can be used as the equivalent of a fixed-point set of arithmetic operations. When so used, the exponent field provides an automatic means of recording scale changes during the course of the program. The unnormalized fraction provides a 48-bit or 96-bit fixed point data field.

Sign Control

The effective sign of an operand used during instruction execution can be determined in three ways. Two operation modifiers, bit positions 19 and 20 of the instruction, are associated with the floating-point operations to provide flexible sign control: The third way is through the operation code, as in ADD TO MAGNITUDE, and the discussion of this action will be found under the various operations.

Instruction bit position 19 is called the "absolute sign modifier." It corresponds to the unsigned modifier in integer arithmetic. The absolute sign modifier applies to the storage operand when the data are fetched from storage. In store operations, the modifier applies to the number which is placed in storage. When the modifier bit is zero, the sign of the operand is used as it exists or is subject to further modification by the negative sign modifier. When the modifier bit is one, the sign of the operand is considered to be positive and the original sign is ignored. This assumed positive sign is subject to further modification by the negative sign modifier.

Instruction bit position 20 is called the "negative sign modifier." It corresponds to the integer arithmetic sign modifier with the same name. The negative sign modifier applies to the operand that is not changed in the operation. In operations that change the accumulator contents, it applies to the operand from storage. In operations that change storage, it applies to the operand from the accumulator. In compare operations, this modifier applies to the operand from storage. When this modifier bit is zero, the sign of the operand, as modified by the operation code or absolute sign modifier, is used unchanged. When this modifier bit is one, the sign of the operand, as modified by the operation code or absolute sign modifier, is inverted. The negative sign modifier, when one, has the effect of changing algebraic additions to subtractions or changing the sign of the result of multiplications, divisions or stores, including the result of STORE ROOT.

When the two sign modifiers are applied to the same operand during an operation, the absolute sign modifier is applied first. However, in some operations, each modifier may be applied to a different operand. These cases are specified in detail in "Operations."

In some operations, the operation code specification causes the accumulator operand to be treated as a positive absolute value while the absolute sign modifier is applied to the operand specified by the effective address. When the operation code itself includes specification of the sign modification, this modification is completed prior to the modification specified by the negative sign modifier.

The sign modifiers always affect the operand signs as they are used. The original operand sign is never changed except when the entire operand is replaced by the result. The use of the data flag bits is not affected by the sign modifiers.

Data Flag Bits

In some problems it may be desirable to mark some of the data to indicate the requirement for special handling. For example, a particular value might belong to a boundary point in a mesh-type calculation. Or, the data might be marked because loss of significance has occurred. In the program it may be desired to attach more than one mark to each data word and treat the marks selectively. The flag bits of the sign bytes associated with integer and floating point data permit such indicative marking. The indicators associated with data flags permit selective action when these flag bits are encountered.

Bit positions 61-63 of a floating-point word are used as data flag bits. These three bits are designated as the T, U, and V bits, respectively. When a floating-point operand is entered into the accumulator by means of the operations LOAD WITH FLAG or LOAD DOUBLE WITH FLAG, the operand flag bits replace the accumulator flag bits in the sign byte register bit positions 5-7. The operations LOAD and LOAD DOUBLE set the data flag bit positions of the sign byte register to zero. All other floating point operations leave the accumulator flag bits unchanged, unless address 10 is used in the to-storage operations. Flag bits of an addressed operand are changed only by operations of the store type, including STORE ROOT, which replace the flag bits in storage by the accumulator flag bits.

The flag bits of a floating-point data word replace the contents of the data flag indicators, TF, UF, and VF, for every operation involving a data fetch. This includes all floating point operations except stores, SHIFT FRACTION and ADD EXPONENT IMMEDIATE.

Zero Definition

A floating-point number having a zero fraction may be interpreted in two ways. It may represent an XFN type of zero or it may represent an order of magnitude. When it is considered as an XFN zero, the exponent and sign are immaterial. When it is considered as an order of magnitude, the exponent and fraction sign indicate the approximate magnitude and sign of the quantity. Such a quantity may result from the cancellation of two non-zero numbers, each representing only a limited amount of accuracy. The result of this cancellation is not to be a valid zero and, therefore, the exponent and sign of the result fraction are meaningful. The discussion of zeros in this section considers only zeros of the order of magnitude type. XFN zeros are discussed in "Range Definition and Handling."

Floating-point values with zero fractions and exponent fields with EF $=$ 0 are called "order of magnitude zeros." In single-precision operations, the high-order 48 bits and the overflow bit of the 96-bit intermediate result fraction are tested for zero. In double-precision operations, all 96 bits and the overflow bit of the intermediate result fraction are tested for zero. In either case, if the tested bits are zero, post-normalization, if specified, is suspended. The intermediate result exponent field, the zero fraction, and fraction sign become the final result value. A zero fraction is not normalized and cannot be normalized.

The fraction sign of an order of magnitude zero resulting from addition is the fraction sign of the operand originally having the alegbraically larger exponent, as modified by the operand and modifiers. When both operand exponent magnitudes and signs are equal, the fraction sign of the order of magnitude zero sum is the modified fraction sign of the operand in the accumulator. The fraction sign of an order of magnitude zero resulting from an operation other than addition is determined from the rules of algebra. Computation of the exponent for a multiplication, division, or square root operation is not changed when magnitude-zero result fraction is obtained. In these cases, the intermediate result exponent becomes the final result exponent without further modification.

The add magnitude operations replace the sum of an addition with a zero fraction when a fraction sign reversal is obtained during the addition operation. This type of order of magnitude zero is called a "forced zero." The fraction sign of the forced zero is the unmodified original fraction sign of the accumulator in ADD TO MAGNITUDE and the unmodified original fraction of the addressed operand in ADD MAGNITUDE TO MEMORY. A forced zero turns on the result zero indicator, RZ, but not the lost significance indicator, LS.

The RZ indicator is set for all operations with a zero fraction result. The LS indicator is set when a zero results from cancellations or shifting. In additions, LS may be set only when at least one operand is not zero.

In comparisons, order-of-magnitude zeros are considered equal only when their exponents differ by 48 or less. When a divisor has a zero fraction, the ZD indicator is set. Division is not executed and accumulator and storage contents remain unchanged.

To leave a negative zero exponent in the accumulator as a result of a floating point operation, a negative zero exponent must be in the accumulator when a divide by zero is attempted (divisor fraction zero). If so, the ZD indicator is set and division is ended.

COMPARE or COMPARE FOR RANGE leaves a negative zero exponent in the accumulator if the following instructions are unnormalized and specify no sign modification (a minus 0 exponent is loaded into the accumulator): FLOATING POINT LOAD, LOAD DOUBLE, LOAD DOUBLE WITH FLAG, and LOAD WITH FLAG. A minus 0 exponent can be stored by an unnormalized FLOATING POINT STORE which specifies no sign modification.

Range Definition and Handling

As outlined in "Data Format," three classes of exponent ranges are provided. These are classified XFP, N, and XFN, and are defined as follows:

RANGE	EF	ES	DEFINITION
XFP	1	0	$\text{Exponent} \geq +1024$
N	0	0 or 1	$+1024 > \text{Exponent} > -1024$
XFN	1	1	$-1024 \geq \text{Exponent}$

Result values, determined by floating point operations, may have flagged exponents from two distinct occurrences. The first way a result may fall in the XFP or XFN ranges is through normal overflow or underflow produced by the action of operands in the N range. The second way that a flagged exponent appears in the result is through forced action associated with the use of operands having flagged exponents.

An exponent flag is said to be *generated* whenever original operands, with EF of 0, develop a result having EF of 1, through overflow or underflow from the N range. An exponent flag is said to be *propagated* whenever an original operand, with EF of 1, forced the result to have EF of 1. For arithmetic operations, when one or more operands are in the XFP or XFN range, the result always has a propagated exponent flag of 1, except for addition, where one operand is in the N range and the other is in the XFN range. In such additions the result may pass from the N range to the XFN range through post-normalization, and a generated EF of 1 is developed.

Figure 23 summarizes the result exponent ranges for the initial operand ranges as specified. The underlined values represent arbitrary choices selected to propagate extreme values in the most noticeable range. The result values in the N range, shown as N*, may be values in the XFP or XFN ranges owing to normal exponent overflow or underflow.

When the result contains a *propagated* EF of 1, normalization, if specified, will be suppressed. When the result contains a *generated* EF of 1, normalization will usually occur, if specified.

When two operands originally with EF of 0 generate an EF of 1 and the subsequent normalization could produce a double carry, the normalization will

ADDITION TABLE

		Addend	
Augend	XFP	N	XFN
XFP	$\underline{\text{XFP}}$	XFP	XFP
N	XFP	N*	N*
XFN	XFP	N*	XFN

MULTIPLICATION TABLE

		Multiplicand	
Multiplier	XFP	N	XFN
XFP	XFP	XFP	$\underline{\text{XFP}}$
N	XFP	N*	XFN
XFN	$\underline{\text{XFP}}$	XFN	XFN

DIVISION TABLE

		Dividend	
Divisor	XFP	N	XFN
XFP	$\underline{\text{XFP}}$	XFN	XFN
N	XFP	N*	XFN
XFN	XFP	XFP	$\underline{\text{XFP}}$

Figure 23. Result Exponent Ranges

be suppressed. These cases occur in multiplication of values with large EM and the same ES, and in division where the values have large EM and different ES. This definition insures that overflowed or underflowed values will remain in that range.

For comparison operations, Figure 24 shows the result relative to the accumulator. Comparisons are made algebraically so that the effective signs of the operand fractions are the primary consideration. A value with a positive fraction sign is considered larger than a value with a negative fraction sign, except that plus zero is equal to negative zero under certain conditions. When the values have the same fraction sign, comparison is based on the relative magnitudes of the exponents and exponent signs and, finally, on the fraction magnitudes. An asterisk indicates a result which further depends on the relative magnitudes of the fractions.

Noisy Mode

The normalization process used with floating-point operations introduces zeros into the low-order bit positions of the fraction as left shifting occurs. This process may result in a loss of significance throughout the course of a program. Assistance in the study of such effects has been provided through an alternative method of operation called the "noisy mode." Noisy mode operation provides for the introduction of ones, rather than zeros, in the low-order bit positions during the left shift associated with normalization. Processing the program both in the standard mode and in the noisy mode provides an estimate for the study of significance loss due to fraction truncation.

The choice of standard or noisy mode operation is specified by the setting of the noisy-mode indicator, NM. When the indicator is zero, standard operation is used; when it is set to one, noisy mode operation will be used. Noisy mode is used only when normalized operation is also specified. Unnormalized floating-point operations are not affected by the noisy mode. Therefore, many incidental operations associated with a calculation may be performed without changing the status of the NM indicator, yet still using the higher speed floating-point instructions.

	Storage					
Accumulator	XFP, +	N, +	XFN, +	XFN, -	N, -	XFP, -
XFP, +	=	High	High	High	High	High
N, +	Low	*	High	High	High	High
XFN, +	Low	Low	=	High	High	High
XFN, -	Low	Low	Low	=	High	High
N, -	Low	Low	Low	Low	*	High
XFP, -	Low	Low	Low	Low	Low	=

Figure 24. Comparison Results

With normalized operation specified, noisy mode operation differs from standard operation at the start of execution and during final normalization. At the start of execution in noisy mode, single-precision 48-bit fractions are extended with 48 ones. The operands in all single-precision operations, except MULTIPLY and STORE ROOT, are extended. In additions, only the operand with the greater exponent is extended with ones. (For operands with equal exponents the accumulator operand is extended.) In divisions, both the divisor and dividend are extended for the normalization preceding the actual division. The divisor is truncated to 48 bits for the division operation. The effect of the extension on addends (or augends) and dividends is to provide one bits instead of the zero bits normally provided when forming the 96-bit intermediate operand. The effect of this extension is to provide low order one-bits on left shifts in normalizing of 48-bit fractions for divisors and other operands. When the final normalization is specified, the extended ones are truncated so that only 48 bits appear in the single-precision results.

At the start of execution for double-precision operations, all operands are assumed to be 96 bits in length. Only those operands requiring pre-execution normalization will acquire one-bits rather than zero bits in the process. The operand in LOAD FACTOR and the divisor in DIVIDE DOUBLE are extended. After the subsequent normalization they are truncated to 48-bit operand fractions. When the final normalization is specified, the 96-bit intermediate result, when shifted left, is extended on the right with ones. The operand in SHIFT FRACTION is not affected with noisy mode bits.

Indicators

The indicators that may be turned on by floating point operations can be divided into two groups. The first group of indicators is of a general nature and can be turned on by variable field-length operations as well as floating-point operations. The description here is with respect to the floating point operations only. The second group of indicators are specifically associated only with floating-point operations.

Indicators are either permanent or temporary. The permanent indicators are turned on by certain specified operation conditions. Once on, they stay on until turned off by an interruption or by programming. The temporary indicators are made either zero or one, depending on the causing condition. These indicators are changed when a new operation is performed which sets the indicators. The temporary indicators remain unchanged for operations with which they are not associated.

Indicators Set by VFL or Floating-Point Operations

Lost Carry (LC). This indicator is turned on when a fraction overflow bit occurs in an unnormalized addition or when a low order one-bit of a remainder is lost in unnormalized DIVIDE DOUBLE. The indicator is also turned on during the operation SHIFT FRACTION, when a one is lost because of a left shift past accumulator bit position 12. This indicator is permanent.

Partial Field (PF). This indicator is turned on in unnormalized division when the magnitude of the dividend fraction is larger than or equal to the magnitude of the divisor fraction. The indicator is permanent.

Zero Divisor (ZD). This indicator is turned on during a division when the divisor fraction is zero. The indicator is permanent.

Data Flag T (TF); Data Flag U (UF); Data Flag V (VF). These indicators are collectively referred to as the data flag indicators. They are set according to the flag bits (T, U, and V) in the sign byte of the floating point words which are obtained from storage. The indicators are temporary.

To-Memory Operation (MOP). This indicator is set according to the operation type of the floating point instruction. The indicator is set to one for all operations of the to-storage type, i.e., those for which the result appears in the location specified by the effective address. Such operations consist of the stores and add to storage operations. For all other floating point operations, the indicator is set to zero. The indicator is temporary.

Result Less than Zero (RLZ); Result Zero (RZ); Result Greater than Zero (RGZ); Result Negative (RN). These indicators are collectively referred to as the arithmetic result indicators. They are set according to the sign and magnitude of the fraction result of any floating point operation except the comparison operations. The indicators are temporary.

The indicator RZ is turned on for all zero result fractions. The indicator RZ will not be turned on when unnormalized operations result in an overflow bit and zero fraction. In this case only the lost carry indicator, LC, will be affected.

Accumulator Low (AL); Accumulator Equal (AE); Accumulator High (AH). These indicators are collectively referred to as the comparison result indicators. They are set according to the result of the floating point comparisons. They indicate the comparative value of the accumulator as compared with the storage operand. The indicators are temporary.

Indicators Set Only by Floating Point Operations

Imaginary Root (IR). This indicator is turned on if the operand of a STORE ROOT operation is negative. The indicator is permanent.

Lost Significance (LS). This indicator is turned on if the result fraction of a floating point addition or shifting operation is zero unless the following exception conditions exist. This indicator is not turned on if both operands in a floating point addition have zero fractions prior to the pre-addition shifting, if the zero fraction is a forced zero as in ADD TO MAGNITUDE, or if the result has a propagated exponent flag. This indicator is permanent.

Preparatory Shift Greater than 48 (PSH). This indicator is turned on if the exponent difference of operands in floating point addition, both having EF of 0, is more than 48. It is turned on for both single- and double-precision addition. It is turned on if the intermediate product in MULTIPLY AND ADD has a *generated* EF of 1 and its use during the addition leads to an exponent difference of more than 48. This indicator is not turned on if either operand in a floating point addition is in the XFP range, or if both operands are values in the XFN range. The indicator is permanent.

Exponent Flag Positive (XPFP) *Exponent* $\geqq 1024$

This indicator is turned on if the result exponent has a propagated EF of 1 and ES of 0. The indicator is permanent.

Exponent Overflow (XPO) *Exponent* $\geqq 1024$

This indicator is turned on if the result exponent has a generated EF of 1 and ES of 0. The indicator is permanent.

Exponent Range High (XPH)

$+1024 > \text{Exponent} \geqq +512$

This indicator is turned on if the result exponent magnitude has a high order 1, and EF of 0, ES of 0. The indicator is permanent.

Exponent Range Low (XPL)

$+512 > \text{Exponent} \geqq +64$

This indicator is turned on if the result exponent magnitude has a high-order zero and a 1 in bit positions 2, 3, or 4 and EF of 0, ES of 0. This indicator is permanent.

Exponent Underflow (XPU)   Exponent ≦ — 1024

This indicator is turned on if the result exponent has a generated EF of 1 and ES of 1. It is permanent.

These indicators are collectively referred to as the exponent range indicators. They permit monitoring of the result exponents of floating point operations. In the case of the overflow and underflow indicators, the occurrence of an incorrect result, because of a lost exponent carry, is signalled. However, the exponent overflow bit sets EF to 1.

The range high indicator signals results which use all available bit positions in the exponent magnitude. Continued computation on these numbers may result in overflow conditions.

The exponent range indicators collectively permit monitoring of computations whose operands and results are below the respective magnitude ranges shown.

The exponent range indicators are set according to the final result of all floating point operations except comparisons. The indicators are permanent.

Zero Multiply (ZM). This indicator is turned on if normalized multiplication or the final result of a normalized multiply and add has an exponent greater than -1024 and a zero fraction. This indicator is temporary.

Remainder Underflow (RU)   Exponent ≦ — 1024

When the remainder exponent in DIVIDE DOUBLE has a generated EF of 1 and ES of 1, this indicator is turned on. It is not turned on when the remainder exponent has a propagated EF of 1. The indicator is permanent.

Noisy Mode (NM). This indicator is on when computations are performed in the noisy mode. The indicator must be set to zero or one by programming. In contrast to other indicators, the noisy mode indicator is never turned on or off automatically. Therefore, it is permanent, except as set by programming.

The operation DIVIDE DOUBLE is the only floating point operation which has two results, a quotient and a remainder. Most of the above indicators apply to the primary result, the quotient. The indicators which may be turned on to describe the remainder are lost carry (LC) and remainder underflow (RU).

In the detailed description of the operations, only the applicable indicators are listed.

Operations

Floating-point operations are divided into single-precision and double-precision operations. The following table lists the two groups and the location of the result and the arithmetic action.

SINGLE-PRECISION OPERATIONS		RESULT ALTERS	ARITHMETIC ACTION
+	Add	Accumulator	Add
+MG	Add to Magnitude	Accumulator	Add
L	Load	Accumulator	Replace
LWF	Load with Flag	Accumulator	Replace
M+	Add to Memory	Storage	Add
M+MG	Add Magnitude to Memory	Storage	Add
ST	Store	Storage	Replace
K	Compare	Indicators only	Add
KMG	Compare Magnitude	Indicators only	Add
KR	Compare for Range	Indicators only	Add
KMGR	Compare Magnitude for Range	Indicators only	Add
*	Multiply	Accumulator	Multiply
/	Divide	Accumulator	Divide
R/	Reciprocal Divide	Accumulator	Divide
SRT	Store Root	Storage	Square Root
DOUBLE-PRECISION OPERATIONS			
D+	Add Double	Accumulator	Add
D+MG	Add Double to Magnitude	Accumulator	Add
DL	Load Double	Accumulator	Replace
DLWF	Load Double with Flag	Accumulator	Replace
SRD	Store Rounded	Storage	Add
SLO	Store Low Order	Storage	Replace
D*	Multiply Double	Accumulator	Multiply
LFT	Load Factor	FT Register	Replace
*+	Multiply and Add	Accumulator	Multiply, Add
D/	Divide Double	Accumulator, RM Register	Divide
F+	Add to Fraction	Accumulator	Add
SHF	Shift Fraction	Accumulator	Shift
E+	Add to Exponent	Accumulator	Add
E+I	Add Immediate to Exponent	Accumulator	Add

Add (+)

The operand specified by the effective address is algebraically added to the operand in the left half of the accumulator. The exponent and fraction of the sum replace accumulator bits 0-59. Accumulator bits 60-127 remain unchanged. The fraction sign of the addressed operand is modified by the sign modifiers prior to the addition. The sign of the sum replaces the accumulator sign, bit position 4, of the accumulator sign byte register. Bits 0-3 and 5-7 of the accumulator sign byte register remain unchanged.

The addition of floating point numbers consists of an exponent comparison and a fraction addition. The exponent of the addressed operand is subtracted from the exponent of the operand in the accumulator. The fraction of the number with the algebraically smaller exponent is shifted right by the amount of the exponent difference. The exponent of the operand with the algebraically larger exponent is used as the exponent of the intermediate result prior to postnormalization. The fraction sign of the addressed operand is modified by the sign modifiers prior to the addition. The fractions are added after the value with the lower exponent has been shifted right by the amount of the difference in exponents. A 96-bit intermediate sum fraction, and possible overflow bit, is formed by extending each operand fraction to 96-bit length by the addition of low-order zeros. The low-order bits of the fraction shifted to the right are retained in the intermediate 96-bit fraction and participate in the addition. When the difference in exponents exceeds 48, indicator preparatory shift greater than 48 (PSH) is set to one, and the low-order bits of the fraction having the low exponent which are shifted beyond the right end of the 96-bit intermediate sum do not take part in the operation. When the difference in exponents exceeds 95, no addition is performed and the result is the number with the higher exponent.

When unnormalized operation is specified, the 48 high-order bits of the 96-bit intermediate sum replace accumulator bits 12-59. The overflow bit does not enter the accumulator, but turns on indicator lost carry (LC).

When normalized operation is specified, the entire 96-bit intermediate sum fraction and overflow bit are shifted to form a normalized fraction and the result exponent is adjusted accordingly. This normalization is not performed if the overflow bit and high-order 48 bits of the intermediate sum are zero. The 48 high-order bits of the normalized sum fraction replace accumulator bits 12-59.

When the noisy mode and normalized operation are specified, the 48-bit fraction of the number with high exponent is extended with 48 low-order one-bits. If the operands have equal exponents, the accumulator operand is extended. The extension takes place before the intermediate sum is formed. Post-normalization occurs as described above.

If either or both operands are values in the XFP or XFN range, initially having an exponent flag, EF of 1, the operation is performed as follows. Modification of the fraction sign of the addressed operand is made as specified by the sign modifiers. If the operands have both exponents in the XFP range or both in the XFN range, the result is the operand in the same range having the algebraically larger exponent. When both operands have the same exponent, the operand initially in the accumulator is the result. Normalization, if specified, is suppressed. If one operand has an exponent in the XFP range and the other has an exponent in the N or XFN range, the result is the operand in the XFP range. Normalization, if specified, is suppressed. If one operand is in the N range and the other is in the XFN range, the intermediate result is the operand in the N range. In this case, however, normalization, if specified, is performed. This may lead to a generated EF of 1 through underflow.

INDICATORS

Data Flags (TF, UF, VF). These indicators are set according to the flag bits of the addressed operand. These indicators are temporary.

To-Memory Operation (MOP). This indicator is set to zero. This indicator is temporary.

Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set according to the final arithmetic result fraction which appears in the accumulator. These indicators are temporary.

Lost Carry (LC). This indicator is turned on if a fraction overflow occurred in an unnormalized addition. The indicator is permanent.

Lost Significance (LS). This indicator is turned on in addition when at least one operand fraction is nonzero if the high-order 48 bits of the intermediate sum and overflow bit are zero and the intermediate result exponent flag is 0. In this case, the final sum fraction is zero and a normal order of magnitude zero is produced with no further normalization, if specified, or exponent change.

This indicator will not be turned on in addition if both operands had zero fractions at the start of the operation or if the intermediate result exponent flag has a propagated EF of 1. This indicator is permanent.

Preparatory Shift Greater than 48 (PSH). When both operands are values in the N range and the algebraic difference of the exponents is greater than 48, or if one operand is a value in the N range while the other operand is a value in the XFN range, this indicator is turned on. When the operands are both values in the XFP range or both in the XFN range, or when one operand is a value in the XFP range while the other is a value in the N or XFN ranges, the result has a propagated exponent flag of 1, and this indicator is not turned on. This indicator is permanent.

Exponent Range (XPFP, XPO, XPH, XPL, XPU). These indicators are set according to the exponent range of the final result appearing in the accumulator. When the result has a propagated EF of 1, the XPFP is set. The ZM will be turned off if on as a result of a previous operation.

PROGRAMMING NOTE

For most arithmetic operations, if one of the operands has EF of 1, the result has EF of 1 and the flag is said to be propagated. In addition operations, however, a value in the N range, EF of 0, added to a value in the XFN range, EF of 1, produces an intermediate sum with exponent in the N range. If this result exponent is altered during normalization, when specified, the exponent flag may be set to 1 in the final result. When this occurs, the exponent flag produced is said to be a generated exponent flag.

Add to Magnitude (+MG)

This operation is performed in the same manner as ADD, with the following exceptions:

The fraction sign of the operand in the accumulator is assumed positive and the result fraction is made a forced zero when a negative sum is obtained.

When the fraction sign of the sum is positive, the sum is placed in accumulator bits 0-59 and the accumulator sign remains unchanged. When the fraction sign of the sum is negative, a zero fraction is placed in accumulator bits 12-59. The sign and exponent of the accumulator operand become the sign and exponent of the forced order of magnitude zero result. For this forced zero, the indicator lost significance (LS) is not turned on, distinguishing it from a zero sum. In an ADD TO MAGNITUDE operation, the sign byte register bit position 4 never changes.

When either or both operands lie in the XFP or XFN ranges, the operation is performed as described in ADD. The fraction sign of the accumulator operand is assumed positive and the result fraction is made a forced zero when a negative sign is obtained. As described above, the sign and exponent of the accumulator operand become the sign and exponent of the forced order of magnitude result.

INDICATORS

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Lost Carry (LC); Preparatory Shift Greater than 48 (PSH); Exponent Range (XPFP, XPO, XPH, XPL, XPU; Zero Multiply (ZM). These indicators are set as in ADD.

Lost Significance (LS). This indicator is set as in ADD, with the additional exception that if the intermediate sum is negative, which gives as a final result a forced zero fraction, it is not turned on.

PROGRAMMING NOTE

The ADD TO MAGNITUDE operation allows arithmetic to be performed on the magnitude of the accumulator contents. Since the arithmetic action of the accumulator operand is restricted to positive magnitude, sign change is not allowed; instead, a forced zero replaces the result.

Load (L)

The operand specified by the effective address is placed in the left half of the accumulator. The exponent and fraction of the operand replace accumulator bits 0-59. Accumulator bits 60-127 are not changed. The sign of the operand, as changed by the sign modifiers, replaces the accumulator sign, bit position 4 in the accumulator sign byte register. The accumulator flag bits are set to zero. When normalized operation is specified, the result in the left half of the accumulator is normalized and zero bits enter the low-order positions. If the address operand has a zero fraction, normalization is suppressed because the result is an order of magnitude zero. If noisy mode is also specified, ones enter the low-order fraction bits during normalization.

When the specified operand is a value in the XFP or XFN range, normalization, if specified, is suppressed. The propagated EF of 1 causes exponent range indicator, XPFP, to be set on. When the EF of 1 is generated by normalization, the exponent range indicator XPU is turned on.

INDICATORS

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Exponent Range (XPFP, XPH, XPL, XPU); Zero Multiply (ZM). These indicators are set as in ADD.

Load with Flag (LWF)

This operation is the same as LOAD except that the data flag bits T, U, and V of the addressed operand sign bytes set the flag bits in positions 5-7 of the accumulator sign byte register as well as the data flag indicators TF, UF, and VF.

INDICATORS

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Exponent Range (XPFP, XPH, XPL, XPU); Zero Multiply (ZM). These indicators are set as in ADD.

PROGRAMMING NOTE

The operations LOAD, LOAD WITH FLAG, LOAD DOUBLE and LOAD DOUBLE WITH FLAG are the only floating point operations which affect the data flag positions in the accumulator sign byte register. No floating point operation changes the zone bits, bit positions 0-3, of the accumulator sign byte register, unless address 10 is used as the addressed operand in a to-storage operation.

Add to Memory (M+)

The operand in the left half of the accumulator is added to the operand specified by the effective address. The exponent and fraction of the sum replace bits 0-59 of the storage word. When normalized operation is specified, the sum is normalized before it is placed in storage. The sign of the addressed operand is used as modified by the absolute sign modifier only. The negative sign modifier is used to modify the sign of the accumulator. The sign of the sum is placed in bit position 60 of the storage word. The flag bits, 61-63, of the storage word remain unchanged. The contents of the entire accumulator and its sign byte register remain unchanged throughout the operation, unless these registers are used as the addressed operand.

The addition of the two operands is identical to the addition described in ADD. The intermediate result is a 96-bit fraction and a possible overflow bit. When normalized operation is specified, the sum is shifted so that the 48 high-order bits of the sum, including the overflow bit, are placed in storage bits 12-59. The exponent of the sum is adjusted by the amount of the shift.

When unnormalized operation is specified, the 48 high-order bits of the sum are placed in storage bits 12-59. The overflow bit does not enter the storage word, but turns on the indicator lost carry (LC).

When noisy mode and normalized operation are specified, the fraction of the operand associated with the larger exponents, or the accumulator fraction if the exponents are equal, is extended with 48 low-order ones before the sum is formed. The result fraction is truncated to 48 bits prior to storing in storage.

If either or both of the operands lie in the XFP or XFN range, the operands are added, after modification of the fraction signs as stated above, in accordance with the rules specified under ADD. When the result has a propagated EF of 1, normalization, if specified, is suppressed.

INDICATORS

Data Flags (TF, UF, VF). These indicators are set according to the flag bits of the addressed operand. These indicators are temporary.

To-Memory Operation (MOP). This indicator is set to one. This indicator is temporary.

Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set according to the final arithmetic result fraction which appears in storage. These indicators are temporary.

Lost Carry (LC). This indicator is turned on if a fraction overflow occurred in unnormalized addition. This indicator is permanent.

Lost Significance (LS). This indicator is set on when at least one operand fraction is non-zero if the high-order 48 bits of the intermediate sum and overflow bit are zero and the intermediate result exponent flag is zero. The final sum fraction is zero, and a normal order of magnitude zero is formed with no further normalization or exponent change. This indicator is not turned on when both operand fractions are zero originally, or if the intermediate result exponent flag has a propagated EF of 1. This indicator is permanent.

Preparatory Shift Greater than 48 (PSH). When both operands are values in the N range, or if one operand is in the N range while the other operand is a value in the XFN range and the algebraic difference of exponents is greater than 48, this indicator is turned on. When both operands are values in the XFP or XFN ranges or when one operand is a value in the XFP range while the other is a value in the N or XFN range, the result has a propagated EF of 1 and this indicator is not turned on. This indicator is permanent.

Exponent Range (XPFP, XPO, XPH, XPL, XPU). These indicators are set according to the exponent range of the final result appearing in storage. When this result has a propagated EF of 1, XPFN will be set. When this result has a generated EF of 1, XPO or XPU will be set.

Zero Multiply (ZM). If on, this indicator will be turned off.

PROGRAMMING NOTE

In contrast to unsigned ADD TO MEMORY in integer arithmetic, this operation causes the result sign to replace the sign of the storage operand. This constitutes the essential difference between unsigned operation, as in integer arithmetic, and absolute operation as in floating point.

Add Magnitude to Memory (M+MG)

Same operation as ADD TO MEMORY except the accumulator operand sign is assumed positive before the negative sign modification occurs, and the sum fraction is replaced by forced zeros when its sign differs from the sign of the addressed operand developed after its modification by the absolute sign modifier.

The sum sign is compared with the addressed operand effective sign. With equal signs, the sum replaces the addressed storage operand. With opposite signs, the result fraction is a forced zero in storage. The forced zero sign is the same as the addressed operands original sign. The sign and exponent of the storage operand become the sign and exponent of the forced order-of-magnitude zero result. For a forced zero, the lost significance (LS) indicator is not set. This action distinguishes it from a zero sum.

INDICATORS

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Lost Carry (LC); Preparatory Shift Greater than 48 (PSH); Exponent Range (XPFP, XPO, XPH, XPL, XPU); Zero Multiply (ZM). The above indicators are set as in ADD TO MEMORY.

Lost Significance (LS). This indicator is set as in ADD TO MEMORY except that a forced zero does not turn it on.

PROGRAMMING NOTES

The ADD MAGNITUDE TO MEMORY operation allows the storage operand, or its magnitude, to be altered as in

ADD TO MEMORY, with the constraint that it cannot be changed in sign. Where a sign change would occur in ADD TO MEMORY, a forced zero occurs in ADD MAGNITUDE TO MEMORY. Furthermore, this operation uses only the magnitude of the accumulator contents instead of the signed accumulator contents used by ADD TO MEMORY.

In contrast to the operation ADD TO MAGNITUDE, where the accumulator operand is assumed always positive, the ADD MAGNITUDE TO MEMORY can operate with positive or negative operands in storage.

The table of arithmetic actions shown in Figure 13 is reproduced in Figure 25 for floating point operations and operands. Results which do not correspond to the integer arithmetic results are underlined. The mnemonic symbols N and A refer to the sign modifiers.

The differences between Figures 13 and 25 are due to a difference in the unsigned integer storage field and the absolute floating point storage field. In the integer case, no sign is available. The result must be a magnitude only, and the lost carry indicator signals a negative sign. In the floating point case, a sign is always available, even though it may be ignored and a negative sign can be recorded as such.

Store (ST)

The operand in the left half of the accumulator is placed in the location specified by the effective address. The exponent and fraction of the operand are taken from accumulator bits 0-59 and replace bits 0-59 of the storage word. The accumulator sign, sign byte register bit position 4, as modified by the sign modifiers, and the accumulator flag bits, sign byte register bits 5-7, replace bits 60-63 of the storage word. The entire accumulator and its sign byte register remain unchanged throughout the operation unless address 8 is the effective address.

When normalized operation is specified, the 48-bit operand fraction in the accumulator is normalized before being placed in storage. If the noisy mode is also specified, ones enter the low-order fraction bit positions as they are shifted left on normalization. Normalization is suppressed when the 48 fraction bits are zero.

When the exponent flag is 1, in the accumulator operand, normalization, if specified, is suppressed and the exponent range indicators are set in accordance with the rules for propagated EF of 1. If an EF of 1 is generated during normalization, the exponent range indicator XPU is turned on. Thus, for values with initially flagged exponent, the value in storage is the same value as appears in the accumulator.

INDICATORS

To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Exponent Range (XPFP, XPH, XPL, XPU); Zero Multiply (ZM). These indicators are set as in ADD TO MEMORY.

Compare (K)

The operand in the left half of the accumulator is compared with the operand specified in the effective address. The action of the specified sign modifiers on the addressed operand is completed before the comparison is made.

The two numbers are compared algebraically by subtraction. However, the operands in storage and the accumulator are not altered. In the intermediate stage the subtraction operation is identical to ADD except for a sign inversion. The result is discarded and the comparison result indicators rather than the arithmetic result indicators are set.

When the exponent difference between the operands exceeds 48, the algebraically larger of the two numbers, judged by exponent and fraction sign, is considered high. This statement is true when one or both of the numbers are zero. When the exponent difference is 48 or less, a zero result is considered an equality, and for this case + 0 is equal to — 0.

In an ADD operation, the normalization modifier specifies post-normalization. Since there is no stored arithmetic result in a comparison operation, the normalization modifier affects the operation only during noisy mode operation. When noisy mode and normalized operation are specified, the operand having the higher exponent, or the accumulator operand, if both operands have equal exponents, is extended with 48 low-order ones prior to forming the intermediate difference, as in ADD.

Storage Operand ⟶	3+	5+	3+	5+	3-	5-	3-	5-
Accumulator Operand ⟶	5+	3+	5-	3-	5+	3+	5-	3-
↓ Operation and Modifiers								
Add	8+	8+	2-	2+	2+	2-	8-	8-
Add, N	2+	2-	8-	8-	8+	8+	2-	2+
Add, A	8+	8+	2-	2+	8+	8+	2-	2+
Add, N, A	2+	2-	8-	8-	2+	2-	8-	8-
Add to Magnitude	8+	8+	8-	8-	2+	0+	2-	0-
Add to Magnitude, N	2+	0+	2-	0-	8+	8+	8-	8-
Add to Magnitude, A	8+	8+	8-	8-	8+	8+	8-	8-
Add to Magnitude, N, A	2+	0+	2-	0-	2+	0+	2-	0-
Add to Memory	8+	8+	2-	2+	2+	2-	8-	8-
Add to Memory, N	2-	2+	8+	8+	8-	8-	2+	2-
Add to Memory, A	8+	8+	$\underline{\text{2-}}$	2+	$\underline{\text{8+}}$	$\underline{\text{8+}}$	$\underline{\text{2-}}$	$\underline{\text{2+}}$
Add to Memory, N, A	$\underline{\text{2-}}$	2+	8+	8+	$\underline{\text{2-}}$	$\underline{\text{2+}}$	$\underline{\text{8+}}$	$\underline{\text{8+}}$
Add Magnitude to Memory	8+	8+	8+	8+	0-	2-	0-	2-
Add Magnitude to Memory, N	0+	2+	0+	2+	8-	8-	8-	8-
Add Magnitude to Memory, A	8+	8+	8+	8+	8-	8-	8-	8-
Add Magnitude to Memory, N, A	0+	2+	0+	2+	0-	2-	0-	2-

Figure 25. Floating Point Operation Results

When one or more of the operands are values in the XFP or XFN ranges, comparisons are performed as follows. When the fractional signs are unlike, the value having the positive fraction sign is considered greater. When the fraction signs are alike, the exponent range is considered. The following criteria apply. If both operands have fraction signs that are positive, a value in the XFP range is considered greater than a value in the N or XFN range, and a value in the N range is considered greater than a value in the XFN range. If both operands have fraction signs that are negative, a value in the XFN range is considered greater than a value in the N or XFP range, and a value in the N range is considered greater than a value in the XFP range. If both operands are in the XFP range or both in the XFN range and both fraction signs are alike, the comparison is considered equal.

The arithmetic result and exponent range indicators and the original contents of the accumulator and of storage are not changed.

INDICATORS

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Preparatory Shift Greater than 48 (PSH). These indicators are set as in ADD.

Comparison Result (AL, AE, AH). Set according to result of the comparison. These indicators refer to the operand in the accumulator as compared with the storage operand.

PROGRAMMING NOTE

Figure 26 outlines the combinations of elements (operand signs, operand magnitudes, subtraction results) that determine the compare indicator settings shown. These elements are: Fa — accumulator fraction sign; Ea — accumulator exponent sign; Fm — storage fraction sign after negative and absolute sign modification; Em — storage exponent sign; Ae — accumulator exponent; Me — storage exponent; Af — accumulator fraction; Mf — storage fraction; and Rs — sign of the discarded result as determined by the modified fraction, signs, and the fraction subtraction result. Results shown are produced for the * values shown in Figure 27.

Figure 27 shows the indicator settings associated with flagged exponent operands. The settings are given relative to the accumulator operand. Column and row values give the exponent range and fraction sign for the respective operands.

	Fa	+	+	+	+	+	+	+	+	-	-	-	-	-	-	-	-
	Ea	+	+	+	+	-	-	-	-	-	-	-	-	+	+	+	+
	Fm	+	+	-	-	+	+	-	-	+	+	-	-	+	+	-	-
	Em	+	-	-	+	+	-	-	+	+	-	-	+	+	-	-	+
Preshift			AH	AH	AH	AL		AH	AH	AL	AL		AH	AL	AL	AL	
>	$\lvert Ac\rvert > \lvert Me\rvert$	AH					AL					AH					AL
48	$\lvert Ac\rvert < \lvert Me\rvert$	AL					AH					AL					AH
Preshift	Af-Mf=0	AE	AE	AE	AE	AE	AE	AE	AE	AE	AE	AE	AE	AE	AE	AE	AE
≤ 48	Af-Mf≠0 Rs (+)	AH	AH	AH	AH	AH	AH	AH	AH			AL	AL			AL	AL
	Af-Mf≠0 Rs (-)	AL	AL			AL	AL			AL	AL	AH	AH	AL	AL	AH	AH

Figure 26. Indicator Settings, Operands

	Storage					
Accumulator	XFP, +	N, +	XFN, +	XFN, -	N, -	XFP, -
XFP, +	AE	AH	AH	AH	AH	AH
N, +	AL	*	AH	AH	AH	AH
XFN, +	AL	AL	AE	AH	AH	AH
XFN, -	AL	AL	AL	AE	AH	AH
N, -	AL	AL	AL	AL	*	AH
XFP, -	AL	AL	AL	AL	AL	AE

Figure 27. Indicator Settings, Flagged Operands

Compare Magnitude (KMG)

The comparison is performed as described for COMPARE except that the comparison is performed as if the sign of the accumulator were positive.

INDICATORS

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Comparison Result (AL, AE, AH); Preparatory Shift Greater than 48 (PSH). The indicators above are set as in COMPARE.

Compare for Range (KR)

When the indicator accumulator high (AH) is on, a comparison will be performed between the operand in the left half of the accumulator and the operand specified by the effective address. If the indicator accumulator high (AH) is off, the comparison is performed but the comparison indicators are not changed.

INDICATORS

Data Flags (TF, UF, VF). These indicators are set according to the flag bits of the addressed operand whether or not the comparison is performed.

To-Memory Operation (MOP). This indicator is set to zero whether or not the comparison is performed.

Accumulator Low (AL). This indicator is not changed.

Accumulator Equal (AE). This indicator is set to one if the accumulator operand is low when compared with the addressed operand.

Preparatory Shift Greater than 48 (PSH). This indicator is set as in ADD.

Accumulator High (AH). This indicator remains one if the accumulator operand is equal or high when compared with the addressed operand. This indicator is set to zero when the accumulator operand is low compared with the addressed operand.

PROGRAMMING NOTE

Following a COMPARE operation, the operation COMPARE FOR RANGE can be used to determine whether a quantity falls within a given range. The addressed operand of the COMPARE operation may be considered as the lower bound of the range. The addressed operand of the COMPARE FOR RANGE operation may be considered as the upper bound of the range.

When both operations have been performed, the settings of the comparison result indicators are interpreted as follows:

Accumulator low (AL) is one when the operand in the accumulator is below the range.

Accumulator equal (AE) is one when the operand in the accumulator is within the range or equal to the lower boundary.

Accumulator high (AH) is one when the operand in the accumulator is above the range or equal to the upper boundary.

The lower boundary is included in the range while the upper boundary is presumed to be outside the range.

Compare Magnitude for Range (KMGR)

This operation is the same as COMPARE FOR RANGE except that the comparison is performed as if the sign of the accumulator were positive.

INDICATORS

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Comparison Result (AL, AE, AH); Preparatory Shift Greater than 48 (PSH). These indicators are set as in COMPARE FOR RANGE.

Multiply (*)

The operand specified by the effective address, the multiplicand, is multiplied by the operand in the left half of the accumulator, the multiplier. The product exponent and fraction replace accumulator bits 0-59. Accumulator bits 60-127 remain unchanged. The sign of the product replaces the accumulator sign in bit position 4 of the sign byte register.

Multiplication of floating point numbers consists of an exponent addition and a fraction multiplication. The sum of the exponents is used as the exponent of the unnormalized result. The two 48-bit operand fractions are multiplied to form a 96-bit intermediate product. The product sign is determined by the rules of algebra, using the accumulator sign and the effective sign of the addressed operand. The sign of the addressed operand is modified according to the specifications of the sign modifiers.

If a normalized product is specified, the 96-bit intermediate product fraction is normalized. The product exponent may be diminished by a value up to 96. Following the normalization, the intermediate product fraction is truncated to 48 bits to form the result product fraction that is placed in accumulator bits 12-59. Low-order zeros are inserted in the low-order positions of the 96-bit intermediate product fraction as normalization occurs. If noisy mode is also specified, low-order ones instead of zeros will be inserted during normalization. The zeros or ones do not appear in the result product fraction unless the amount of post-normalization shift is greater than 48.

When unnormalized operation is specified, the 48 high-order bits of the intermediate product fraction are used as the result product fraction and are placed in accumulator bits 12-59.

For values of the operands in the XFP or XFN range, the following action is taken. When both operands are in the XFP range or both in the XFN range, the result exponent is the exponent of the accumulator operand. The result fraction is the unnormalized product of the operand fractions. If one operand is in the XFP range and the other is in the N or XFN range, the result exponent is the exponent of the operand in the XFP range and the result fraction is the unnormalized product of the operand fractions. If one of the operands is in the N range while the second is in the XFN range, the result exponent is the exponent of the operand in the XFN range and the result is the unnormalized product of the operand fractions. For all these cases normalization, if specified, is suppressed. Thus, whenever either operand in a multiplication has an EF of 1, the result exponent always has a propagated EF of 1.

When both operands are values in the N range, a generated EF of 1 may be produced because of overflow or underflow. With underflow, when EF of 1 and ES of 1 are part of the result exponent, normalization of the result fraction, if specified, is suppressed in order to prevent occurrence of a double carry. However, the indicator settings follow the rules for generated exponent flags.

INDICATORS

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Exponent Range (XPFP, XPO, XPH, XPL, XPU). These indicators are set for the final arithmetic result as in ADD.

Zero Multiply (ZM). If normalized operation is specified and an order of magnitude zero results with exponent greater than -1024, this indicator will be turned on. Otherwise it will be turned off, if already on.

Divide (/)

The operand in the left half of the accumulator, the dividend, is divided by the addressed operand, the divisor. The sign of the addressed operand is modified by the sign modifiers prior to the division. The quotient exponent and fraction replace accumulator bits 0-59. Accumulator bits 60-127 remain unchanged. The quotient sign replaces the accumulator sign in bit position 4 of the sign byte register. No remainder is retained by this operation.

In both the normalized and the unnormalized mode, divisions will be performed whether the operands are initially normalized or not. Since this requirement sometimes conflicts with the rule that unnormalized results are not shifted, the indicator partial field (PF) is used to identify the case, in unnormalized operations, in which the quotient is shifted right to preserve high-order bits, and the left zeros counter, LZC, is used to indicate the amount of the shift. The only exception to the rule that all divisions are performed, is the case of a zero fraction in the divisor, for which division is suppressed. Indicator zero divisor (ZD) is used to signal the occurrence of a zero divisor fraction. When division is suppressed because of a zero divisor, the accumulator and the sign byte register remain unchanged. The data flag and MOP indicators will be set but the arithmetic result and exponent range indicators are not changed. With normalized operation (when division is performed), a normalized quotient is placed in the accumulator.

In the case of unnormalized operation, the quotient placed in the accumulator may or may not be normalized. When the dividend fraction, developed after pre-division normalization, is smaller than half the divisor fraction, developed after pre-division normalization, leading zeros will occur in the unnormalized quotient fraction. The quotient exponent, for operands in the N range initially, is equal to the difference of the dividend and divisor exponents when the magnitude of the divisor fraction is greater than the magnitude of the dividend fraction after pre-division normalization is completed. When the magnitude of the divisor fraction is equal to or smaller than the magnitude of the dividend fraction after pre-division normalization, an overflow condition exists but a correct shifted quotient is developed.

Details of Operation

The operation described below applies when both operands are in the N range at the start of the operation.

Pre-division normalization is applied to both the divisor and dividend prior to the actual division. Prenormalization of the divisor is independent of the normalization modifier. The normalization process extends the divisor with zeros during left shifting. The amount of left shift is subtracted from the divisor exponent and set in LZC as a positive quantity. If the fraction is zero, normalization is suppressed and the indicator zero divisor (ZD) is set to one. When the incator ZD is set to one, further processing of the division is suspended and the accumulator contents remain undisturbed. If a normalized quotient is specified and noisy mode is also specified, the divisor normalization extension is made with ones instead of zeros inserted in the low-order positions.

Subsequent to the divisor normalization, the dividend fraction, extended with 48 low-order bits to form a 96-bit fraction, may be shifted left. When unnormalized operation is specified, the amount of the shift is equal to the shift that occurred in divisor normalization, or the left shift required for dividend normalization, whichever is less. In either case, the amount of the shift is subtracted from the dividend exponent and from the left zeros counter, LZC. A zero value in the LZC indicates that both operands were shifted the same amount.

When normalized operation is specified, the dividend is fully normalized. The value in the LZC will remain zero if the left shift for dividend normalization exceeds the amount required for divisor normalization. If normalized operation and noisy mode are specified, the extension to 96 bits is made with 48 ones instead of zeros and ones are introduced in the low-order bits of the 96-bit dividend fraction during normalization. If the dividend fraction is zero, the amount of left shift is only the amount specified by the value in the LZC.

In the division process, as a result of pre-division normalization, the divisor is always normalized and the dividend may or may not be normalized. Both operands have adjusted exponents and the LZC contains a value indicating the excess, if any, of divisor shifting over dividend shifting. The divisor has a 48-bit fraction and the dividend, a 96-bit fraction.

The dividend exponent adjusted for pre-division normalization and quotient overflow, if required, minus the adjusted divisor exponent, is the initial intermediate quotient exponent. The sign of the quotient is determined by the rules of algebra using the accumulator sign and the effective sign of the addressed operand.

If the magnitude of the divisor fraction is greater than the magnitude of the dividend fraction, 48 quotient bits are obtained. This quotient fraction may or not may not be normalized depending on the completeness of the dividend normalization and the relative magnitudes of the operand fractions.

If the magnitude of the divisor fraction is less than or equal to the magnitude of the dividend fraction, a quotient overflow condition is produced and corrected. The correction is accomplished by adding one to the adjusted intermediate dividend exponent and placing a one in the high-order bit position of the quotient fraction. Subsequently, 47 more quotient fraction bits are obtained. Also a one is added to the value of the LZC.

The final result of the operation above is to produce a fully normalized quotient fraction when normalization is specified, except for conditions outlined below for operands in the XFP or XFN ranges. The result quotient exponent and fraction are placed in accumulator bits 0-59. The quotient sign is placed in bit position 4 of the sign byte register. If unnormalized operation was specified and the value in the LZC is greater than zero, the indicator partial field (PF) is set to one. With unnormalized operation the result quotient may or may not be normalized.

The contents of LZC remain available for program use until the LZC is changed by integer multiplication, LOAD TRANSIT, division or connective operations.

Operand pre-normalization may cause an operand to go from the N range to the XFN range. Division proceeds as described above except for the case when one operand is in the XFN range and the other is in the N range with ES of 0. For this case, if the divisor has a generated value in the XFN range, the quotient exponent has EF of 1, ES of 0 and EM of the exponent magnitude of the divisor. If the dividend has the generated value in the XFN range, the quotient exponent is the same as the dividend exponent. For both situations the quotient fraction is developed normally and indicator settings are set in accordance with the rules for generated exponent flags.

When either or both of the operands are in the XFP or XFN range prior to pre-normalization, the pre-normalization process follows the rules for unnormalized processing. This may result in unnormalized quotient fractions. If the divisor fraction is zero, the operation is terminated regardless of the exponent and the zero divisor indicator is turned on. When the divisor fraction is non-zero, exponent handling follows the following rules.

Divisor: Exponent in XFP or N range
Dividend: Exponent in the XFP range

The quotient exponent is the same as the exponent of the dividend. Thus the quotient lies in the XFP range.

Divisor: Exponent in XFP range
Dividend: Exponent in N or XFN range

The quotient exponent is the same as the divisor exponent with the exponent sign reversed. Thus the quotient lies in the XFN range.

Divisor: Exponent in N range
Dividend: Exponent in the XFN range

The quotient exponent is the same as the dividend exponent. Thus the quotient lies in the XFN range.

Divisor: Exponent in the XFN range
Dividend: Exponent in the XFP, N or XFN range

The quotient exponent is the divisor exponent with the exponent sign reversed. Thus the quotient lies in the XFP range.

When either operand in division has EF of 1, the exponent of the result always has a propagated EF of 1.

INDICATORS

Data Flags (TF, UF, VF). These indicators are set according to the flag bits of the addressed operand. They are temporary.

To-Memory Operation (MOP). This indicator is set to zero. It is temporary.

Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set according to the quotient fraction as it appears in the accumulator. They are temporary.

Partial Field (PF). This indicator is turned on when unnormalized operation is specified, if the left zeros counter (LZC) is greater than zero.

Zero Division (ZD). This indicator is turned on when the divisor fraction is zero. In this event, the division does not take place. This indicator is permanent.

Exponent Range (XPFP, XPO, XPH, XPL, XPU). These indicators are set according to the exponent range of the quotient, and are permanent.

Zero Multiply (ZM). If on, this indicator will be turned off, except for the case of a zero divisor.

Reciprocal Divide (R/)

The operand specified by the effective address, the dividend, is divided by the operand in the left half of the accumulator, the divisor. The sign of the dividend is modified by the sign modifiers prior to the division. The quotient exponent and fraction replace accumulator bits 0-59. Accumulator bits 60-127 remain unchanged. The sign of the quotient replaces the accumulator sign in bit position 4 of the sign byte register.

The division process is the same as the process described in DIVIDE with the exception of the interchange of the operands used for dividends and divisor.

INDICATORS

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Partial Field (PF); Zero Divisor (ZD); Exponent Range (XPFP, XPO, XPH, XPL, XPU); Zero Multiply (ZM). These indicators are set as in DIVIDE.

Store Root (SRT)

The square root of the operand in the left half of the accumulator is placed in the storage location specified by the effective address. The entire accumulator and its sign byte remain unchanged.

For values of the operand in the N range the square root operation consists of an exponent division and a fraction square root operation. The exponent is divided by two. When the exponent is odd, a one is added to the exponent prior to the division and the fraction is shifted right one position. Subsequently, the square root of the 48- or 49-bit fraction is obtained. During the process the fraction is extended with low-order zeros. The result is a 48-bit fraction.

When normalized operation is specified, the result is normalized and placed in storage. When unnormalized operation is specified, the result is placed unchanged in storage. In both cases the result is placed in bit positions 0-59 of the storage location specified by the effective address. No noisy mode 1-bits are entered when normalizing and noisy mode are specified.

The absolute sign modifier is applied to the accumulator sign prior to the extraction of the square root. When this modifier is zero, the accumulator sign is used unchanged. If the accumulator sign is negative, the imaginary root indicator (IR) is turned on. When the absolute sign modifier is one, the accumulator sign is assumed positive and IR cannot be turned on. The negative sign modifier is applied to the result of the square root process as in the store operation. When this modifier is zero, the sign of the stored result is positive. When this modifier is one, the sign of the stored result is negative.

As with other store operations, the accumulator flag bits, sign byte register bits 5-7, are placed in the flag bit positions, bits 61-63 of the addressed storage location.

When the operand is a value in the XFP or XFN range, the result exponent is the same as the operand exponent. The result fraction is the square root of the accumulator fraction. Normalization, if specified, is suppressed. Note that when the operand exponent is odd the fraction is shifted one position to the right prior to taking the square root. However, the exponent is not modified as a result of the shift for operands in the XFP or XFN range.

INDICATORS

To-Memory Operation (MOP). This indicator is set to one. It is temporary.

Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set according to the arithmetic result which appears in storage. They are temporary.

Imaginary Root (IR). This indicator is turned on when the accumulator sign, as modified by the absolute sign modifier, is negative. It is permanent.

Exponent Range (XPFP, XPH, XPL). These indicators are set according to the exponent range of the result which appears in storage. These indicators are permanent.

Zero Multiply (ZM). This indicator will be turned off, if on.

PROGRAMMING NOTE

The negative sign modifier permits selection of the sign to be attached to the root. The absolute sign modifier governs whether the sign of the radicand is to be checked.

Add Double (D+)

The operand specified by the effective address is added to the operand in the accumulator bits 0-107. The addressed operand is extended with 48 zeros during the operation. The exponent and fraction of the sum replace the accumulator bits 0-107. Accumulator bits 108-127 remain unchanged. The sign of the addressed operand is modified by the sign modifiers prior to the addition. The sign of the sum replaces the accumulator sign.

The detailed operation of the addition is similar to that of ADD. When the difference between the exponents exceeds 48, preparatory shift greater than 48 indicator, PSH, is turned on. When the difference between the exponents exceeds 95 the number with the lower exponent is not added. The result is then the number with the higher exponent. The fraction addition yields a 96-bit sum and possible fraction overflow bit.

When normalized operation is specified, the sum fraction is normalized and replaces bits 12-107. If noisy mode is also specified, one bits enter the low-order bit positions when the 96-bit intermediate result is shifted left during normalization. Noisy mode one bits are not added to either operand prior to the addition as is done in ADD.

When unnormalized operation is specified, the 96-bit intermediate sum fraction is placed in accumulator bits 12-107. The overflow bit does not enter the accumulator, but turns on the lost carry indicator.

If the 96-bit intermediate sum has a zero fraction, normalization, if specified, will be suppressed. The exponent replacing accumulator bits 0-11 will be the exponent of the intermediate result. The lost significance indicator (LS) is set to one.

When the operand values lie in the XFP or XFN range, the addition process follows the operation outlined under ADD except that the accumulator fraction is 96 bits and the operand specified by the effective address is extended with 48 zeros prior to the addition.

INDICATORS

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Lost Carry (LC); Preparatory Shift Greater than 48 XPU); Zero Multiply (ZM). These indicators are set as in ADD.

Lost Significance (LS). This indicator is set to one when at least one operand is non-zero and when the 96 bits of the result fraction are zero. This indicator is not set to one if the result exponent has a propagated flag or if both operands have 96 bit zero fractions initially.

Add Double to Magnitude (D+MG)

This operation is the same as ADD DOUBLE except that the sign of the accumulator operand is assumed positive and the result is made a forced zero when a negative sum is obtained.

When the sign of the sum is positive, the sum is placed in the accumulator bits 0-107 and the sign remains unchanged. When the sign of the sum is negative, a zero fraction is placed into bits 12-107 of the accumulator and the sign and exponent of the accumulator operand remain equal to the original accumulator sign and exponent.

Normalization, noisy mode, and values in the XFP or XFN range are handled as in ADD DOUBLE.

INDICATORS

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Lost Carry (LC); Preparatory Shift Greater than 48 (PSH); Exponent Range (XPFP, XPO, XPH, XPL, XPU); Zero Multiply (ZM). These indicators are set as in ADD TO MAGNITUDE.

Lost Significance (LS). This indicator is actuated as in ADD DOUBLE except that a forced zero does not turn it on.

Load Double (DL)

This operation is the same as LOAD except that the accumulator bits 60-107 are set to zero.

The operand specified by the effective address is placed in the accumulator. The exponent and fraction replace bits 0-59. Accumulator bits 60-107 are set to zero. Accumulator bits 108-127 remain unchanged. The sign of the operand, modified by the sign modifiers, is placed in bit position 4 of the sign byte register. The flag bits, bit positions 5-7 of the sign byte register, are set to zero.

When normalized operation is specified, the 96-bit fraction is shifted left and the exponent is adjusted before being placed in the accumulator. If noisy mode is also specified, one bits are introduced into the low-order positions of the 96-bit fraction as it is shifted left during normalization.

Values in the XFP or XFN ranges are handled as in LOAD. Normalization of these operands, if specified, is suppressed.

INDICATORS

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Exponent Range (XPFP, XPH, XPL, XPU); Zero Multiply (ZM). The above indicators are set as in ADD.

Load Double with Flag (DLWF)

The operation is the same as LOAD DOUBLE except that the flag bits T, U, and V of the addressed operand set the accumulator flag bits in the sign byte register bit positions 5-7 as well as the data flag indicators TF, UF, VF.

INDICATORS

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Exponent Range (XPFP, XPH, XPL, XPU); Zero Multiply (ZM). The above indicators are set as in ADD.

Store Rounded (SRD)

The contents of the accumulator after rounding are placed in the location specified by the effective address.

The rounding is performed by adding one to the fraction in the accumulator in bit position 60. This addition results in a 96-bit sum and a possible overflow bit. The 48 low-order bits are discarded. When normalized operation is specified, the 48-bit sum is then normalized and placed in bit positions 12-59. Low-order zeros are always supplied in the case of the left shift. Noisy mode does not affect this operation.

When unnormalized operation is specified, the 48 high-order bits of the sum are placed in storage bits 12-59. The overflow bit does not enter storage but sets the lost carry indicator (LC) on.

The sign of the accumulator, as changed by the sign modifiers, is placed in storage bit 60. The accumulator flag bits are placed in storage bits 61-63. The entire accumulator and its sign byte register remain unchanged throughout the operation unless used as the effective address.

If the value of the operand lies in the XFP or XFN range, the rounding process is performed, but the exponent is not changed. Normalization, if specified, is suppressed. If, as a result of rounding, the fraction has an overflow bit set, the overflow bit is lost. However, the lost carry indicator (LC) is not set if overflow occurs when normalization is specified.

INDICATORS

To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Lost Carry (LC); Exponent Range (XPFP, XPO, XPH, XPL, XPU); Zero Multiply (ZM). These indicators are set as in ADD TO MEMORY.

Store Low Order (SLO)

The low-order part of a double-length accumulator fraction with appropriate exponent is placed in the location specified by the effective address.

The low-order bits of the accumulator fraction, bits 60-107, become the fraction of the value to be stored. The exponent is obtained by subtracting 48 from the exponent in bits 0-11 of the accumulator.

When normalized operation is specified, the low-order fraction is shifted left until a high-order one bit is leftmost. If noisy mode is also specified, ones are entered from the right during the normalization shifting; otherwise, zeros. When unnormalized operation is specified, the low-order exponent and fraction remain unchanged. The exponent and fraction are subsequently placed in storage bits 0-59. The accumulator sign, as modified by the sign modifiers, and the accumulator flag bits, sign byte register bit positions 4-7, are placed in bits 60-63 of the storage word. The entire accumulator and sign byte register remain unchanged throughout the operation unless used as the effective address.

When the exponent, accumulator bit positions 0-11, is flagged, the same exponent is set with the low-order fraction when stored. The indicator settings are for propagated flags. For this case normalization, if specified, is suppressed.

When the exponent with the low order fraction has EF of 1 as a result of the operation, the indicator settings are for generated exponent flags. Normalization, if specified in this case, does occur.

Indicators

To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Exponent Range (XPFP, XPH, XPL, XPU); Zero Multiply (ZM). The above indicators are set as in STORE.

Programming Example

Given two double-precision floating point numbers, A and B stored at the following locations:

$$C(100) = a_1$$
$$C(101) = a_2$$
$$C(102) = b_1$$
$$C(103) = b_2$$

Here $A = a_1 + a_2$ and exponent a_1 = exponent $a_2 + 48$; $B = b_1 + b_2$ and exponent b_1 = exponent $b_2 + 48$.

Form the double precision difference $A - B$ and store it in locations 104 and 105 (Figure 28).

NAME	STATEMENT	NOTES
	DL (U), 101	
	D- (N), 103	1
	D+ (N), 100	2
	D- (N), 102	3
	ST (U), 104	
	SLO (U), 105	

Notes: 1. Form low-order difference $a_2 - b_2$
2. Add high-order <u>A</u>: $a_1 + (a_2 - b_2)$
3. Subtract high-order <u>B</u>: $(a_1 + a_2 - b_2) - b_1$

Figure 28. Double-Precision Subtract Example

Multiply Double (D*)

The operation is the same as MULTIPLY, except that the 96-bit product fraction is placed in accumulator bits 12-107.

The operand specified by the effective address, the multiplicand, is multiplied by the operand in the accumulator bits 0-59, the multiplier. The product exponent and fraction replace accumulator bits 0-107. The product sign replaces the accumulator sign, bit 4 of the sign byte register.

The detailed operation of the multiplication is similar to that of MULTIPLY. An intermediate product exponent and 96-bit fraction are obtained.

When normalized operation is specified, the 96-bit intermediate product fraction is normalized and the product exponent adjusted accordingly. Low-order fraction zeros are supplied. If the noisy mode is also specified, the final normalization introduces low-order ones instead of zeros. When unnormalized operation is specified, the intermediate product becomes the final product.

When either or both of the operands are values in the XFP or XFN range, the operation is performed as described in MULTIPLY except that a 96-bit fraction is obtained. Normalization, if specified, is suppressed for these cases.

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Range (RLZ, RZ, RGZ, RN); Exponent Range (XPFP, XPO, XPH, XPL, XPU); Zero Multiply (ZM). The above indicators are set as in MULTIPLY.

Programming Note

The exponent range indicators are set for the final result as it appears in the accumulator or in storage. In general, exponent overflow or underflow occurring in an intermediate stage of an operation may be corrected and thus does not cause the XPO or XPU indicators to be turned on.

In order that double underflow carry in the product exponent will not occur during multiplication when both original operands were values in the N range, the following rule is applied. If the product exponent, from operands both in the N range, has EF of 1 and ES of 1, prior to post-normalization, normalization, if specified, is suppressed. The XPU indicator is turned on in accordance with the rules for generated exponent flags. However, if the generated EF is 1 and ES is 0, normalization of the product fraction, if specified, occurs and the XPO or XPH indicators may be turned on in accordance with the final exponent range.

Load Factor (LFT)

The operand specified by the effective address is placed in the factor register, FT. The accumulator does not participate in the operation and remains unchanged.

When normalized operation is specified, the exponent and fraction of the addressed operand are normalized and subsequently are placed in FT bits 0-59. If noisy mode is also specified, ones instead of zeros will enter the low-order bits of the fraction during normalization. If a shift of 48 or more is required, normalization is suppressed and the order of magnitude zero is stored in bits 0-59.

When unnormalized operation is specified, the exponent and fraction of the addressed operand are placed unaltered in FT bits 0-59.

The sign of the addressed operand as modified by the sign modifiers is placed in FT bits 60. FT bits 61-63 are set to zero.

When the operand is a value in the XFP or XFN ranges, normalization, if specified, is suppressed.

INDICATORS

Data Flags (TF, UF, VF). These indicators are set according to the flag bits of the addressed operand.

To-Memory Operation (MOP). This indicator is set to zero.

Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set according to the arithmetic result which appears in the FT register.

Exponent Range (XPFP, XPH, XPL, XPU). These indicators are set according to the exponent range of the result which appears in the FT register.

Zero Multiply (ZM). This indicator will be turned off, if on.

PROGRAMMING NOTE

This operation is required for loading the multiplicand of a subsequent MULTIPLY AND ADD operation. The operation is not used to load either factor of a MULTIPLY operation.

The factor register, FT, is addressable and, therefore, can be addressed as the storage operand of any operation. FT is a 64-bit register with word address 14.

LOAD FACTOR is the only operation which changes the contents of FT as an integral part of its performance. FT is not changed by any other operation unless it is addressed explicitly.

Multiply and Add (*+)

The operand in the FT register, designated as the multiplicand, is multiplied by the operand specified by the effective address, designated as the multiplier. The product is added to the contents of the accumulator, bit positions 0-107. The sum exponent and fraction replace accumulator bits 0-107. The sign of the sum replaces the accumulator sign, sign byte register bit 4. Accumulator bits 108-127 remain unchanged.

The intermediate product consists of an exponent formed from the sum of the operand exponents and a 96-bit fraction formed as the product of the two 48-bit operand fractions. The intermediate product fraction remains unnormalized throughout the addition. The sign of this product is developed by the rules of algebra using the sign of the FT register and the effective sign of the addressed operand as developed through modification of the operand sign by the sign modifiers.

Subsequent to the multiplication, an addition is performed similar to that described in ADD DOUBLE. When the difference in exponents exceeds 48, the indicator preparatory shift greater than 48, PSH, is set to one. When the difference in exponents exceeds 95, the number with the smaller exponent is not added. In this case the result is the number with the high exponent. Addition of two 96-bit fractions yields a 96-bit sum fraction and possibly an overflow bit. When normalized operation is specified, the sum is shifted so that the high-order bits of the sum, including the overflow bit, are placed in the accumulator bit positions 12-107. The sum exponent is adjusted by the amount of the shift. When noisy mode is specified with normalized operation, one bits are entered in the low-order bit positions of the 96-bit sum fraction during the left shifting associated with normalization.

This operation initially specifies three operands. They are the operand in the FT register, the operand specified by the effective address, and the operand in the accumulator. The description above considered the details of operation when all operands were in the N range and all intermediate results remained in the N range. When values of the operands or intermediate results fall in the XFP or XFN ranges, the following action is taken.

During multiplication, the operands used are the operand in the FT register and the operand specified by the effective address. If either operand exponent has EF of 1, the product exponent has a propagated EF of 1. When both operands are in the XFP range or both in the XFN range, the product exponent is the exponent of the operand specified by the effective address. When one operand is in the XFP range while the other is in the N or XFN range, the product exponent is the exponent of the operand in the XFP range. When one operand is in the N range while the other is in the XFN range, the product exponent is the exponent of the operand in the XFN range.

For all cases, the intermediate product fraction is the unnormalized fraction determined as the product of the two operand fractions. The normalization modifier applies only to the final result fraction developed during the addition portion of this operation.

During addition, the operands used are the intermediate product developed during multiplication and the operand in the accumulator. When both operands are in the XFP range or both in the XFN range, the result is the operand, exponent and fraction, in the same range having the algebraically larger exponent. When both operands have the same exponent value, the operand initially in the accumulator is the result. Normalization, if specified, is suppressed. If one operand is in the XFP range while the other is in the N or XFN ranges, the result is the operand, exponent and fraction, in the XFP range. Normalization, if specified, is suppressed. If one operand is in the N range and the other is in the XFN range, the intermediate result sum is the operand in the N range. In this case, normalization, if specified, occurs.

The exponent range indicator is set in accordance with the propagated or generated exponent flag when the sum exponent has EF of 1. The sum, EF of 1, is considered generated if all operands are initially in the N range or if the intermediate product has a value in the XFN range with a propagated exponent flag while the accumulator operand is in the N range but the sum has EF of 1, ES of 1 owing to final normalization. All other cases lead to a propagated EF of 1, in the final result.

INDICATORS

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Lost Carry (LC); Preparatory Shift Greater than 48 (PSH); Exponent Range (XPFP, XPO, XPH, XPL, XPU). The above indicators are set as in ADD DOUBLE.

Zero Multiply (ZM). If the final result, after addition, is an order of magnitude zero with exponent greater than −1024 this indicator will be turned on, otherwise, if on, it will be turned off.

PROGRAMMING EXAMPLE

Given two double precision numbers A and B at locations 100-103, form the double-precision product and store it in locations 106-107 (Figure 29).

$$C\ (100) = a_1;\ C\ (101) = a_2$$
$$C\ (102) = b_1;\ C\ (103) = b_2$$

NAME	STATEMENT	NOTES
	DL (U), 100	
	D* (U), 103	1
	LFT (U), 102	
	*+ (N), 101	2
	*+ (N), 100	3
	ST (U), 106	
	SLO (U), 107.	

Notes: 1. $a_1 b_2$
2. $a_2 b_1 + a_1 b_2$
3. $a_1 b_1 + (a_2 b_1 + a_1 b_2)$

Figure 29. Double-Precision Multiply Example

$A = a_1 + a_2$ with exponent a_1 = exponent $a_2 + 48$.
$B = b_1 + b_2$ with exponent b_1 = exponent $b_2 + 48$.

The product term $a_2\ b_2$, which may result in a difference of one in the low-order bit position of the product, is ignored.

Divide Double (D/)

The operand in the accumulator bits 0-107, the dividend, is divided by the operand specified by the effective address, the divisor. The sign of the divisor is modified by the sign modifiers prior to the division. The quotient exponent and fraction replace accumulator bits 0-60. Accumulator bits 61-107 are set to zero. The sign of the quotient replaces the accumulator sign. Accumulator bits 108-127 remain unchanged. The remainder is placed in the remainder register, RM address 13. The original accumulator sign becomes the sign of the remainder RM register bit 60. The flag bits 61-63 of the remainder register are set to zero.

As noted in DIVIDE, division is always performed regardless of the original state of normalization or the setting of the normalization modifier, unless the divisor fraction is zero. The partial field indicator (PF) indicates the occurrence of a quotient shift in the unnormalized operation; the left zeros counter, LZC, indicates the excess of divisor shift over dividend shift for pre-division normalization. In the case of a zero divisor, the division is not performed and the zero divisor indicator (ZD) is set to one.

Double-precision division is designed to provide 48-bit quotient and remainder fractions from a 96-bit dividend fraction and a 48-bit divisor fraction. Depending upon the relative magnitudes of the dividends and divisor fractions, the remainder may be 48 or 49 bits after a 48-bit quotient is developed. In normalized operation, the entire remainder is normalized and in the case of 49 significant bits, the low-order bit is discarded. In unnormalized operation, the 48 high-order bits of the remainder are stored. In either case, the remainder exponent and fraction are stored in the remainder register, RM, bits 0-59. If the low-order 49th bit dropped in storing is a one, the lost carry indicator (LC) is turned on.

All double-precision results can be stored with rounding as single-precision values, by using the STORE ROUNDED operation. In order to allow the quotient of a double-precision division to be rounded, a 49th quotient fraction bit is developed after the remainder is preserved.

Details of Operation

The details of pre-division normalization and the generation of the first 48 bits of the quotient fraction in normalized and unnormalized division follow the description given for DIVIDE except for the introduction of noisy mode ones into the dividend. In the double-precision handling, pre-division normalization for the dividend is permitted to have up to 96 positions of shift. Ones are introduced in noisy mode operation only in positions vacated at the low-order end of the double length fraction as it is shifted left.

In DIVIDE DOUBLE, the remainder is preserved. When the divisor fraction is equal to or smaller than the 48 high-order bits of the dividend fraction, a remainder of 49 bits is obtained. Otherwise a 48-bit remainder fraction is obtained. The intermediate remainder exponent is obtained by subtracting 48 from the adjusted exponent of the shifted dividend.

When normalized operation is specified, the remainder is normalized and placed in RM bit positions 0-59. In the case of a 49-bit remainder, the low-order remainder bit is lost. When unnormalized operation is specified, the remainder is not normalized, but the 48 high-order remainder fraction bits and the remainder exponent are placed unnormalized in RM bit positions 0-59. Again, if a 49-bit remainder was developed, the low-order bit is lost. Lost carry indicator (LC) is set to one if the lost bit is one.

Before normalization, the remainder is compared with the divisor to obtain a 49th quotient fraction bit. The 49-bit fraction is placed in accumulator bits 12-60. The quotient exponent is placed in accumulator bits 0-11. The sign of the quotient, obtained by the rules of algebra, using the dividend sign and the effective divisor sign, replaces the accumulator sign, sign byte register bit 4. Accumulator bits 61-107 are set to zero. The accumulator flag bits, sign byte register bits 5-7, remain unchanged.

When normalized operation is specified, the final quotient placed in the accumulator is normalized, because of the complete normalization of the dividend during pre-division normalization. When unnormalized operation is specified, the quotient may or may not be normalized, depending on the extent of pre-division normalization shifting of the dividend. If the divisor shift for normalization exceeds the dividend shift for relative normalization or if the quotient had an overflow condition corrected, as in DIVIDE, the contents of the left zeros counter are greater than zero. In unnormalized operation the partial field indicator (PF) is set to one when these conditions exist.

When either or both of the operands are values in the XFP or XFN range, the quotient exponent and quotient fraction are developed as described under DIVIDE except that the dividend fraction is initially bits 12-107 of the accumulator. When either or both of the operands are flagged with EF of 1, prior to pre-division normalization, the quotient exponent always has a propagated EF of 1. For this case divisor pre-division normalization will always be complete. However, dividend pre-division normalization of the fraction follows the procedure for unnormalized operation; i.e., the maximum left-shift permitted is the same amount as the divisor left-shift. However, the partial field indicator (PF) is turned on only if unnormalized operation is specified and the LZC has a value after pre-division normalization greater than one. If normalized operation is specified, PF is not turned on through incomplete dividend shifting.

When the quotient has a propagated EF of 1, the remainder exponent is the same as the original dividend exponent, prior to pre-division normalization. Normalization of the remainder fraction, if specified, is suppressed. When the remainder has a propagated EF of 1 and ES of 1, setting of the remainder underflow indicator (RU) is suppressed.

When both operands are originally in the N range, either or both may acquire exponent flag, EF of 1, prior to the division. The divisor may become a value in the XFN range through pre-division normalization. The dividend may become a value in the XFP range through quotient overflow adjustment or a value in the XFN range through pre-division normalization and quotient overflow adjustment, if required. When a generated EF of 1 is developed, division proceeds normally except for the following cases. If the dividend has a generated EF of 1 and ES of 1 while the divisor has EF of 0 and ES of 0, the quotient exponent is the adjusted dividend exponent. If the divisor has a generated EF of 1 and ES of 0 and the dividend has a generated EF of 0 or 1 and ES of 0, the quotient exponent will be the divisor exponent with ES set to 0. This action is required to avoid double overflow or double underflow. The quotient for all cases just discussed may have a generated EF of 1 and the indicators XPO and XPU will be set in accordance with the rules for generated EF of 1.

When both operands, prior to pre-division normalization, are in the N range, the remainder exponent is the dividend exponent (after pre-division normalization and quotient overflow adjustment) minus 48. Normalization, if specified, occurs. If the final remainder exponent has a generated EF of 1 and ES of 1, the remainder underflow indicator (RU) is turned on.

INDICATORS

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Partial Field (PF); Zero Divisor (ZD); Exponent Range (XPFP, XPO, XPH, XPL, XPU); Zero Multiply (ZM). The above indicators are set as in DIVIDE.

Lost Carry (LC). The indicator is turned on if a 49-bit remainder is obtained, when unnormalized division is specified, and the low-order bit which was one was dropped.

Remainder Underflow (RU). When the remainder exponent has a generated EF of 1 and ES of 1, this indicator is turned on. If the remainder exponent has a propagated EF of 1 and ES of 1, this setting of this indicator is suppressed.

PROGRAMMING NOTES

Since a 49-bit quotient is obtained, the result zero indicator (RZ) is turned on only when all 49 quotient bits are zero.

The four examples in Figure 30 illustrate the results which may be obtained in DIVIDE DOUBLE operations. The examples are for 3-bit fractions for operands in memory and for a 6-bit fraction for operands in the accumulator. The exponent is expressed in decimal to the left of the period.

PROGRAMMING EXAMPLES

Given the two double-precision numbers A and B at location 100-103, form the double-precision quotient of A divided by B and store it at locations 108-109 (Figure 31). A and B are assumed normalized double precision values.

$$A = a_1 + a_2 \text{ where exponent } a_1 = \text{exponent } a_2 + 48$$
$$B = b_1 + b_2 \text{ where exponent } b_1 = \text{exponent } b_2 + 48$$

Example		1	2	3	4
Divisor		0.011	0.011	0.101	0.001
Dividend		0.111 011	0.001 011	0.110 011	0.000 101
Shifted	U, N	-1.110	-1.110	0.101	-2.100
Divisor	nm-N	-1.111	-1.111	0.101	-2.111
Shifted	U	0.111 011	-1.010 110	0.110 011	-2.010 100
Dividend	N	0.111 011	-2.101 100	0.110 011	-3.101 000
	nm-N	0.111 011	-2.101 111	0.110 011	-3.101 111
Quotient	U	2.100 1	0.011 1	1.101 0	0.101 0
	N	2.100 1	-1.111 0	1.101 0	0.101 0
	nm-N	2.100 0	-1.110 1	1.101 0	-1.110 1
Remainder	U	-2.101^L	-4.100	-2.000^L	-5.000
	N	-2.101	-6.100	-5.100	-6.000
	nm-N	-4.110	-5.101	-5.100	-6.101
Left Zero Counter		2^P	0	1^P	0

U: Unnormalized
N: Normalized
nm: Noisy Mode
P: Partial Field indicator is turned on in the unnormalized case
L: Lost Carry indicator is turned on

Figure 30. Divide Double Results

Add to Fraction (F+)

The operation is the same as ADD DOUBLE except that the exponent of the operand in the accumulator is used for both the addressed operand and the operand in the accumulator.

The operand specified by the effective address is added to the operand in the accumulator. The exponent of the addressed operand is ignored. Instead, the exponent of the operand in the accumulator is also used as the exponent of the addressed operand. The sum fraction and exponent replace accumulator bits 0-107. Accumulator bits 108-127 remain unchanged. The sign of the addressed operand is modified by the sign modifiers prior to the addition. The sign of the sum replaces the accumulator sign.

When normalized operation is specified, the result fraction is normalized. If noisy mode is also specified, ones enter in the low-order position instead of zeros.

When the accumulator operand is a value in the XFP or XFN range, the fraction addition is not performed since the operand specified by the effective ad-

NAME	STATEMENT	NOTES
	DL (U), 101	
	D+ (U), 100 'a to accumulator	
	D/ (N), 102 'divide by b_1	1
	ST (N), 108	2
	D*N (N), 103 'quotient times b_2	3
	D+ (N), $RM	4
	D/ (N), 102	5
	D+ (N), 108	6
	ST (N), 108	
	SLO (U), 109	

Notes: The double-precision quotient is evaluated according to the formula:

$$\frac{a}{b} = \frac{a_1 + a_2}{b_1 + b_2} = q_1 + q_2 = q$$

where $q_1 = \dfrac{a_1 + a_2}{b_1} - \dfrac{r_2}{b_1}$

and $q_2 = \dfrac{r_2 - q_1 b_2}{b_1} + \dfrac{r_3}{b_1}$

The evaluation neglects third-order terms. Because of this and the possible occurrence of a 49-bit remainder fraction, the double precision quotient fraction may be accurate to 95 bits only.

1. Divide $(a_1 + a_2)$ by b_1 yielding quotient q_1 used and remainder r_2.
2. Store q_1 (48 bit fraction only).
3. Multiply q_1 (48 bit fraction) by $(-b_2)$.
4. Form $r_2 - q_1 b_2$.
5. Divide $r_2 - q_1$ by b_1 yielding quotient q_2 and remainder r_3.
6. Add q_1 and q_2 to form a standard double-precision number.

Figure 31. Double-Precision Divide Example

dress assumes the exponent of the accumulator operand.

INDICATORS

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Lost Carry (LC); Lost Significance (LS); Exponent Range (XPFP, XPO, XPH, XPL, XPU); Zero Multiply (ZM). The above indicators are set as in ADD DOUBLE.

Shift Fraction (SHF)

The fraction in the accumulator is shifted in the direction and amount specified by the effective address bits 0-11. A left shift is specified if bit 11 is zero, a right shift, if bit 11 is one. The amount of the shift is specified by bits 0-10. Zeros are inserted in the bit positions which are vacated. Only accumulator bits 12-107 participate in the shifting; all other accumulator bits remain unchanged. If a one bit is shifted left beyond accumulator bit position 12, it is lost and the lost carry indicator (LC) is set to one. A one-bit shifted right beyond accumulator bit position 107 is lost, but no indicator is turned on. The sign of the shift bit 11 of the effective address, or shift amount, is subject to modification by the sign modifiers. No change in the exponent field of the accumulator operand is made by this operation.

This operation does not address storage. The effective shift may be obtained by standard indexing procedures. The effective shift is not monitored for address boundaries. The operation is not affected by the normalization modifier. No noisy mode one bits are entered on left shift if noisy mode is specified.

When the accumulator operand is a value in the XFP or XFN range, the operation is performed as described above. The result is the original accumulator exponent and the fraction shifted in the specified direction by the specified amount.

INDICATORS

Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set according to the arithmetic result which appears in the accumulator.

Lost Carry (LC). This indicator is set to one when a one-bit is shifted left beyond accumulator bit position 12.

Exponent Range (XPFP, XPH, XPL). These indicators are set according to the exponent range of the result as it appears in the accumulator.

Zero Multiply (ZM). If on, this indicator will be turned off.

To-Memory Operation (MOP). This indicator is set to zero.

Add to Exponent (E+)

The exponent of the operand specified by the effective address is added to the exponent of the operand in the accumulator. The addition is algebraic and takes into account the signs of both exponents. The sign of the exponent of the addressed operand is subject to modification by the sign modifiers. The fraction and sign of the addressed operands are ignored. The result of the exponent addition is placed in accumulator bits 0-11. When normalized operation is specified, the accumulator exponent and fraction bits 12-107 are normalized. Accumulator bits 108-127 and the sign byte register remain unchanged. If noisy mode is also specified low-order one bits instead of zeros are introduced during normalization. When unnormalized operation is specified, normalization is omitted. The sign modifiers are ignored if either one or both of the exponents are in the XFP or XFN range.

When the exponents of either or both operands lie in the XFP or XFN range, the rules for exponent handling follow the definitions given under MULTIPLY. When either operand has EF of 1, the result will have a propagated EF of 1 and normalization, if specified, is suppressed. In this case the result fraction is the original accumulator fraction. When both operands have EF of 0, the result exponent may have a generated EF of 1. For this case normalization, if specified, occurs if ES is 0 and is suppressed if ES is 1.

The multiplication table is shown in Figure 32.

INDICATORS

Data Flags (TF, UF, VF). These indicators are set according to the flag bits of the addressed operand.

To-Memory Operation (MOP). This indicator is set to zero.

Arithmetic Result (RLZ, RZ, RGZ, RN). These indicators are set according to the entire floating point result which appears in the accumulator.

Exponent Range (XPFP, XPO, XPH, XPL, XPU). These indicators are set according to the exponent range of the results.

Zero Multiply (ZM). If on, this indicator will be turned off.

Add Immediate to Exponent (E+I)

The operation is the same as ADD TO EXPONENT, except that the addend is bits 0-11 of the effective address. This operation does not address storage. The effective address field of the instruction is not monitored for address boundaries.

Multiplier	Multiplicand		
	XFP	N	XFN
XFP	XFP	XFP	$\underline{\text{XFP}}$
N	XFP	N*	XFN
XFN	$\underline{\text{XFP}}$	XFN	XFN

*These values may be in the XFP or XFN range as the result of normal overflow or underflow.

Figure 32. Exponent Multiplication Table

The addition is algebraic, taking into account the signs of exponent operands. The sign of the immediate operand, effective address bit 11, is subject to modification by the sign modifiers.

Indicators

To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Exponent Range (XPFP, XPO, XPH, XPL, XPU); Zero Multiply (ZM). These indicators are set as in ADD EXPONENT.

Programming Note

The instruction set does not contain a double-precision store operation. Either of two interpretations for such an order may be programmed, using the single-precision STORE, which will be equivalent. One interpretation is that the double-precision fraction be normalized, after which the high-order 48-bit fraction and exponent are stored. The normalization of the 96-bit fraction can be accomplished through the use of the operation ADD IMMEDIATE TO EXPONENT with a zero operand, unless the accumulator is a value in the XFP range or XFN range.

The second interpretation is to store both halves of the accumulator fraction. In order to accomplish this the two operations STORE and STORE LOW ORDER are used with the normalization modifier specifying unnormalized operation.

Program Example

Given two integers, A and B, which appear as unnormalized numbers with zero exponent in memory locations 200 and 201; store the integer quotient and remainder of A/B as unnormalized numbers with zero exponent in memory locations 202 and 203 (Figure 33).

NAME	STATEMENT	NOTES
	DL (U), 200	
	SHFR 48	1
	D/ (U), 201	2
	ST (U), 202	
	L (U), CON1	3
	+ (U), RM	4
	E+I (U), +48	5
	ST (U), 203	
CON1	DD (N), +0E +48	

Notes:
1. The integer dividend is shifted right 48 places.
2. A zero divisor will turn on indicator ZD.
3. The accumulator is set to exponent -48 and fraction zero.
4. The remainder is aligned.
5. The remainder exponent is made zero.

Figure 33. Program Example

IBM 7619 Exchange

The IBM 7619 Exchange directs information flow between input-output units and core storage. The exchange relieves the central processing unit of the task of communicating with input-output units and enables processing of data to proceed simultaneously with operation of a number of these units.

The exchange provides a general method of connecting many different kinds of units to the computer system. It contains the common control facilities to be time-shared among the external units, thus keeping these units as simple as possible, yet maintaining fully overlapped operation. The exchange also does the necessary housekeeping of addresses and the assembly or disassembly of information without taking time away from the computer or core storage. The only computer time involved is that needed to start and interlock the operations. The only core storage cycles required during external operations are those needed to transfer the data to or from the final locations in core storage. These cycles are sandwiched between computing operations without interfering with the computer program except for the slight delay involved in the actual core storage references.

A common method of program control applies to all external units. When an input-output instruction is given, the computer executes all address modification. It then sends the addresses and the decoded operation to the exchange, which completes the execution of the instruction by obtaining the operand (control word) from core storage and starting the external unit. This procedure permits the exchange, before it accepts an instruction, to determine from its stored status indication bits whether the unit is ready, and to sandwich into available time periods the extra cycles necessary to start an operation. If the unit is not ready (for instance, if the unit is out of material and waiting to be reloaded), the exchange rejects the instruction and sends a signal to the indicator register.

The computer waits until the exchange has signalled that it has accepted or rejected the instruction. If the exchange is operating near full load, the computer may have to wait some microseconds because it is more flexible in its operation than the external units, most of which cannot wait. It usually does not wait for the external unit to respond or to finish the operation, which may take milliseconds to minutes. The exchange takes over full control and signals the computer when the operation is completed. The computer can in the meantime proceed with the program.

An outline of the logical components of the exchange is shown in Figure 34.

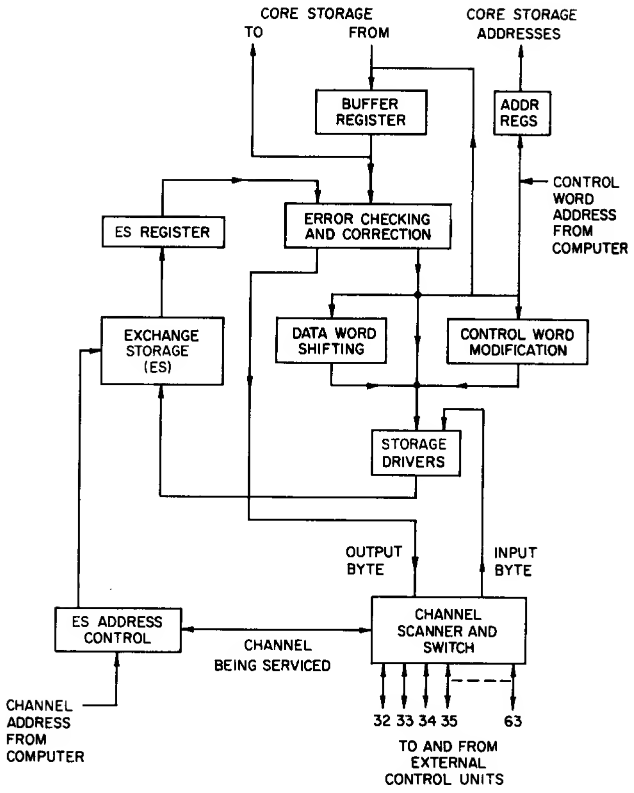


Figure 34. Exchange Components

Capacity

Channels

The exchange provides up to 32 channels for connecting external units directly to the system. These units are of the kind that operate serially one 8-bit byte at a time, and thus require eight bytes to transmit a full storage word of 64 bits. The channels are nine bits wide, accommodating an 8-bit byte and a parity bit to check transmission over the channel. With units that are presently available, a channel can achieve data rates of one byte every 16 microseconds, but higher rates can be accommodated.

The exchange can have several combinations of channels. The minimum is eight. This minimum can be augmented by additional channels in groups of eight up to 24 additional, or a total of 32 channels.

Each exchange channel is connected to one control unit. Most control units are designed to handle only one specific external unit and accommodate only that one unit. Other control units are capable of handling more than one input-output or external storage device of a given type. Only one such device, however, may operate at a time over a single channel, and the program selects which device is connected to the control unit at any one time. Thus, there may be more external devices connected to the system than there are channels, but this does not increase the amount of simultaneous operation.

Simultaneity of Data Transfer

The amount of simultaneous reading and writing depends upon the number of exchange channels and upon the data transmission rates of the units attached. As to the data transmission rates, the limitation can be imposed either by the transmission of bytes of data between the exchange and the units or by the transmission of data words and control words between the exchange and core storage.

The limitations on the simultaneity of data transfer imposed by the transmssion of data between the exchange and the units is expressed by means of the byte weight system. The *byte weight* of a unit indicates the relative amount of exchange capacity which is necessary to service the unit. It depends upon the shortest interval in which the exchange may have to respond to the unit's byte request and is not necessarily proportional to the data transmission rate of the unit.

As far as estimation of the maximum data flow between the exchange and the units is concerned, the external units can be classified into two types: non-buffered units whose operation cannot be interrupted without loss of information, and units which are buffered or because of some other reason can be delayed in case of conflict.

The latter type of units (e.g., card readers, card punches, consoles, printers) are flexible in timing their data transmission and have a byte weight of zero. When the exchange cannot service their byte requests, their operation is interrupted. This interruption, however, is not associated with any loss of information. As soon as the excess load is removed and the exchange has time to spare, the mechanism of the unit is automatically restarted and the original reading or writing operation is continued. The interruption does not have any effect on the completion of the operation, and no indication is given to the computer to this effect.

In the non-buffered type of external units (e.g., tape units), data are transmitted continuously between the external unit and core storage. These units have non-zero weights. When exchange capacity is exceeded, such units cannot interrupt their operation, and, as a result, some data may be lost. When this happens, a unit-check indication is automatically sent to the computer. This indication is identical to that given in case of data errors and can be treated accordingly.

In order to avoid exceeding exchange capacity, the program can keep track of the amount of data flow in process by adding the byte weights of all units operating in the form of a *byte weight count.* The scale of byte weights is defined such that a byte weight count of 256 represents the maximum safe load on the exchange. The total data transmission rate of the exchange corresponding to this byte weight count is not fixed and depends upon the particular set of units used.

Since the byte weight of a unit reflects only the rate of transfer of bytes of data between the exchange and the units, it is fixed for a given type of unit and does not depend upon the mode of operation. Normally the sum of the byte weights indicates when exchange capacity is exceeded. The byte weight system, however, does not reflect the load imposed on the exchange by the transfer of data words for the zero-byte-weight units and the fetching of control words for chained operations. When extensive chaining is employed or many zero-byte-weight units are operated, it is possible that the word transfer between the exchange and the core storage may limit the capacity of the exchange.

In order to facilitate the evaluation of the total instantaneous rate of transmission of information between the exchange and core storage, a *word weight* is defined for each external unit, including the zero-byte-weight units. The word weight of a unit is an indication of the unit's data transmission rate and is equal to 2560 divided by $(T - 11)$, where T is the unit's minimum instantaneous word period expressed in microseconds per word. When a unit is engaged in chained reading or writing, its word weight has to be doubled. The *word weight count,* which is the sum of the word weights of all units reading and writing, then is an indication of the amount of information that is currently being transmitted between the exchange and core storage. As in the case of the byte weight system, the scale of the word weight system is defined such that a word weight count of 256 represents the maximum safe load on the exchange. Whenever the exchange capacity to communicate with core storage is exceeded and a data word or a control word is not transferred, a unit-check indication for that channel is given. The communications between the exchange and core storage limit the maximum instantaneous information transmission rate of the exchange to 100,000 words per second.

Both the above weight systems assume that the units are connected to the exchange in such a way that the unit having the highest byte and word weights has the lowest channel address of the system, with the weights decreasing in the order of ascending channel addresses. If this sequence is violated, a unit's byte and word weights have to be set equal to those of the unit having the largest weights at a higher channel address.

The two weight count systems provide a programming means of keeping the total input-output data transmission rate below the maximum safe limit. When one of the weight counts is exceeded, there is danger of losing some information. However, it should be noted that in specifying the weight systems, the maximum possible instantaneous load on the exchange is considered. In the case of chaining, the doubling of the word weight of a unit affords a further factor of safety in that it permits chaining to occur after each word transmitted to or received from core storage. Thus, on the average, the exchange will not operate as close to capacity as indicated by the weight counts. Unless many channels are installed and units with high data transmission rates are used, in some installations it may not be possible to exceed the exchange capacity at all.

Weight Count Example

Consider a system at an instant when the following units are executing READ or WRITE instructions:

TYPE OF UNIT	NO. OF UNITS OPERATING	CHAINING?	UNIT'S WEIGHT BYTE	UNIT'S WEIGHT WORD	WEIGHT COUNTS BYTE	WEIGHT COUNTS WORD
729 high dens.	2	no	32	17	64	34
729 low dens.	1	yes	9	6	9	12
Card reader	2	no	0	5	0	10
Card punch	1	yes	0	5	0	10
Console	1	no	0	3	0	3
			Total weight counts		73	69

Both the byte and word weight counts are well below the maximum limits, and hence there is no danger of exceeding exchange capacity.

Reading and Writing

Data are transferred between external units and core storage by read and write operations. A read or write instruction is first given by the program. This instruction specifies the external unit (channel address) and the location of a *control word,* which defines the beginning and end of the core storage area to be used for the data transfer. These two pieces of information are sent to the exchange, and the computer then proceeds to the execution of the next instruction in the program.

The control word defines the core storage area to be used in data transfer by specifying the initial data word address in the storage and a count of the number of words to be transferred. After READ or a WRITE is received, the exchange obtains the specified control word from core storage and stores it in the exchange. The desired unit is then started, and reading or writing proceeds, using the storage area indicated by the control word.

As each data word is read or written, the control word in the exchange is modified: the data word address is advanced by one, and the word count is decreased by one. The control word in the exchange always contains the data word address of the next core storage reference and a count of the number of words remaining to be transferred to or from the external unit. The control word in core storage, however, is not affected during the data transfer. Thus, the control word in the core storage unit retains the initial data word address and count, and it can be used repeatedly to transfer data to or from the same storage area.

There are four basic variations of a read or write operation depending on the setting of certain *flag* bits in the control word:

1. The operation may be terminated by the exchange upon reaching a count of zero or by the unit upon reaching the end of the block, whichever happens first. (Single block operation without chaining)
2. After the core storage area defined by a control word is exhausted, it is possible for the exchange to substitute another control word and continue data transfer without stopping, using the core storage area defined by the new control word. (Chaining)
3. More than one block may be read or written with one instruction, until the specified core area has been used. (Multiple block operation)
4. During reading, it is possible to suppress entry into storage of selected portions of the information. (Skipping)

A *block* of data is defined for each type of unit as the amount of information recorded in the interval between adjacent starting and stopping points of the unit. The length of a block depends upon the type of unit used; e.g., it can be a card, a line of printing, or the information between two consecutive gaps on tape.

Control Word Format

The control word format follows that of an index word: the data word address occupies the word address portion of an index word value field while the count and refill fields of control and index words are the same. This facilitates the use of control words for indexing. In addition, the control word contains a number of bits which describe the status of the external units and specify the mode of reading or writing.

The following is a detailed description of the control word format shown in Figure 35.

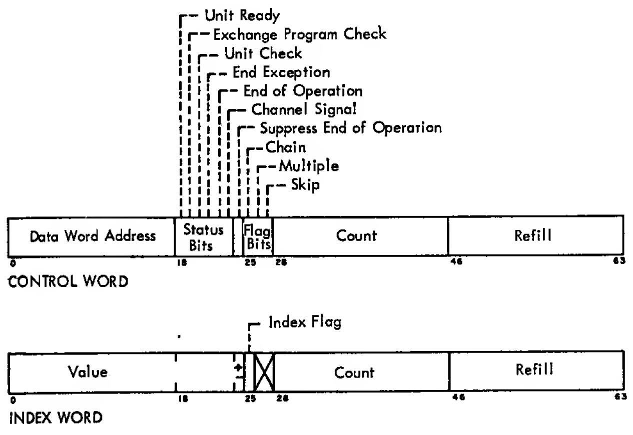


Figure 35. Control Word and Index Word Formats

Bits 0 to 17, Data Word Address

This field contains the address of the first data word in storage. (This field is the word address portion of the value field in an index word.)

Bits 18 to 24, Status Bits

When obtaining a control word from core storage, the exchange ignores these bits and treats them as if they were zero. Inside the exchange, however, these same bit positions are used to retain the status bits which indicate the status of the unit. When the COPY CONTROL WORD instruction is used to extract the control word from the exchange, the status bits appear in the following positions.

18. Unit Ready. This bit is on whenever the unit is in condition to accept an instruction from the computer or is executing an operation. When the bit is off, the unit is in the not-ready status and cannot be operated by the computer.

19-22. Exchange Program Check, Unit Check, End Exception, End of Operation. These status bits signal the cause of the termination of an operation at an external unit.

23. Channel Signal. This bit signals operator request for attention or indicates that a unit has changed from not ready to ready status.

Bits 19 through 23 have corresponding indicators in the indicator register and, when certain conditions are satisfied, cause the indicators to be turned on.

24. Suppress End of Operation. This bit is turned on by a SEOP (suppress end of operation) type instruction to suppress the end of operation indication upon a normal completion of the operation.

Bits 25 to 27, Flag Bits

These are control bits which permit certain variations in reading and writing operations.

25. Chain Flag. If on, another control word is obtained by the exchange when the count in the current control word reaches zero.

26. Multiple Flag. If on, more than one block of data may be read or written with a single instruction.

27. Skip Flag. If on, during reading, transfer of data to storage is suppressed. The skip flag has no effect on writing.

Bits 28 to 45, Count

This field contains the number of words that can be transferred under control of the corresponding control word. (This field has an analogous meaning in an index word.)

Bits 46 to 63, Refill Address

This field contains the address of the next control word to be obtained by the exchange when the count reaches zero and the chain flag is on. (This field has an analogous meaning in an index worα.)

Definition of Core Storage Area

The control word, as the exchange initially obtains it from core storage, contains the address of the first data word in the storage unit. Subsequent data words are sent to or received from consecutive higher core storage addresses, which the exchange obtains by adding one to the data word address in the control word after each data transfer. Similarly, the count of the control word is stepped down by one whenever a word is read or written. If data transfer is not terminated by other means as described below, the process repeats until the cycle at which the count is stepped from one to zero.

Reaching a count of zero requires that the operation be terminated or, if chaining is used, that a new control word be obtained. The number of words that have been transferred is then equal to the number initially specified in the count field. The initial data word address and count thus define the core storage area set aside for this control word. The exchange guarantees that data transfer will not proceed beyond the storage area defined by the control word, even if there is more information in the block being read.

The exchange can only handle and address blocks of information starting at the left end of a full word in core storage and containing an integral number of words (i.e., multiples of 64 data bits). An external unit may, however, terminate an operation at any time within a block or even within a word. In both reading and writing, data transfer takes place a byte at a time, starting with the leftmost byte of a word and proceeding to the right. Consequently, an interruption may prevent the low-order bytes of a word from being read or written. When a reading operation is interrupted within a word, the exchange always completes the word by filling in zero bits to the right of the data bits before sending it to core storage.

The exchange has access to data and control words at any existing core storage address from 32 up. This range does not include internal computer registers and index registers. When the exchange discovers that a control word address or a data word address refers to a core storage location to which the exchange does not have access (locations 0 to 31 or addresses above the maximum available address in a particular installation), the operation is stopped and the exchange-program-check status bit for the appropriate channel is turned on. When the right effective address of an input-output instruction specifies a control word address in the range 0 through 31, the operation is not started, and the address-invalid indicator is turned on. An exception is made during initial program loading, when storage address 4 is used in a special way without giving an error indication.

Chaining

The chain flag of each control word specifies whether this is the last control word or whether another one follows.

If the chain flag is zero, the present control word is the last one, and the refill address is not used. The operation cannot proceed after the count is exhausted. If the chain flag is one, the refill address specifies the address of another control word which the exchange obtains automatically whenever the count reaches zero. In this case, data transfer continues beyond the portion specified by the current control word, unless terminated sooner by other causes.

The chaining technique is used to permit scattering portions of a block of information in different areas of core storage. A "chain" of control words is, in effect, formed in core storage by having each control word specify its successor. The exchange then automatically follows this chain until it encounters a control word with a zero chain flag, or until the external unit stops the process.

Single- and Multiple-Block Operation

The multiple flag specifies whether a new block of data may be started at the end of the current block or not.

When the multiple flag of a control word is zero, reading or writing proceeds either until the unit reaches the end of the block or until the control word reaches a count of zero, whichever happens first. If a new control word in a chained operation is fetched before the end of the block, the multiple flag of the new control word determines the mode of operation during the time the new control word is in force. It follows that if all control words in a chain have the multiple flags set to zero, only a single block of data may be read or written with one instruction. It is possible, however, to read or write less than a block of data by specifying a count which is less than the number of words in the block.

When the multiple flag of a control word is one, operation always proceeds, if not terminated prematurely by exceptional causes, until the number of words transferred equals the number specified in the count field. If the unit meanwhile reaches the end of the block, a new block is automatically started. As before, in a chained operation the multiple flag of the control word in force always determines the mode of operation. With the multiple flag set to one, it thus becomes possible to read or write more than one block of data with one instruction. By letting the multiple flags of all control words in a chained operation be one, the amount of information transferred is completely under control of the control word counts.

When chaining occurs at the end of a block (i.e., when the count of a control word reaches zero upon the transmission of the last data word of a block), the continuation of the operation depends upon the multiple flag of the old control word. Thus, if the chain flag in the old control word is on but the multiple flag is off, the operation is terminated at the end of the block. On the other hand, if both flags are on, the operation is continued regardless of the state of the multiple flag in the new control word.

All signalling between the unit and the exchange is done one block at a time. This applies to the status indications from the unit to the exchange as well as to the control information sent by the exchange to the unit. To obtain multiple block operation, the exchange merely signals the unit to restart as soon as the end of a block is reached, as if a new read or write instruction had been given by the computer for each separate block. The distinction between single- and multiple-block operation is thus retained at the exchange and is not apparent at the unit.

A multiple block operation may, however, be terminated before reaching the last block specified by the instruction. If an exchange program check, a unit check, or an end exception is caused, the operation never proceeds beyond the end of the block in which the condition is discovered.

Whenever the end of a block is reached and a new block is started, data transfer continues with the next full word in core storage. If a block is not an integral number of full words long, on reading, the missing bytes of the last word are filled with zeros. On writing, the bytes remaining after completion of the block are ignored.

The case when both the multiple and the chain flags are on and the control word is exhausted anywhere within a block (i.e., not upon the transmission of the last word of a block) is treated in a special way. When the count reaches zero, the exchange sends a disconnect signal to the unit indicating that it does not expect to transfer any more data to or from the unit until the end of the current block is reached. The unit proceeds upon the receipt of the signal to the end of the current block without any data transfer and without any control word modification. On reading, the remainder, if any, of the current block is skipped. On writing, no more information is sent out until a new block is started. When the end of the block is reached, the unit is restarted, the new control word is fetched and the operation is resumed with the next block. Thereafter, chaining and block control take place in accordance with the chain and multiple flags of the next control word. Thus, in this mode of operation, it is possible to force a gap on tape at the end of any selected control word in a chain.

Successive control words in a chain may have different chain and multiple flag bits. The precise operation may be determined from Figure 36. The exact contents of the control word in the exchange depends upon the type and the stage of the operation. A more detailed description of the exact state of the control word in the exchange is given later.

Multiple Flag	Chain Flag	Count 0 = Zero 1 = Not Zero	End of Block 0 = Yes 1 = No	Action
0	0	0	0	End operation
0	0	0	1	Disconnect, then end op.
0	0	1	0	End operation
0	0	1	1	Continue
0	1	0	0	End operation
0	1	0	1	Chain and continue
0	1	1	0	End operation
0	1	1	1	Continue
1	0	0	0	End operation
1	0	0	1	Disconnect, then end op.
1	0	1	0	Restart and continue
1	0	1	1	Continue
1	1	0	0	Chain, restart and continue
1	1	0	1	Disconnect and chain
1	1	1	0	Restart and continue
1	1	1	1	Continue

Figure 36. Multiple and Chain Flag Operations

Skipping

The skip flag, when zero, specifies normal reading. On writing, the flag is meaningless and is ignored. When the skip flag is one, no data words are transferred to core storage. The operation of the unit is still continued, and the system goes through all the sequences associated with data transfer between the exchange and the unit. The control word is also appropriately modified by incrementing its address and decrementing the count, and, if necessary, a new control word is fetched in chained operations. The assembled data words, however, are not transferred to core storage. By setting the appropriate control words in a chain to skip or not, any selected portion of data may thus be suppressed during reading.

Since the testing for data word addresses that exceed the available core storage is performed in the core storage controls, reference to non-existent core storage locations cannot cause an exchange-program-check indication when reading with the skip flag on. The exchange-program-check indication is always given when an attempt is made to fetch a control word from a core storage address that is not accessible to the exchange.

As a result of the above, the program can specify any address above 31 when reading with the skip flag on. If in the process of control word modification the address 2^{18} is reached, the next cycle resets the address to 0. Since address 0 is invalid, the operation is terminated when an attempt is made to store a word in this location.

The presence of the skip flag does not affect the treatment of the unit-check and end-exception indications. When a unit check or end exception occurs, the operation is terminated in accordance with the general rules.

Chain and Multiple Flag Examples

In Figures 37 and 38, a section of a continuous line bounded by arrow heads indicates data transfer under control of one control word. A dashed line indicates skipping over the remainder of a block without any data transfer and without any control word modification taking place. The latter happens after a disconnect signal on units where the length of the blocks is fixed. The vertical lines at the ends of arrows indicate the start or the termination of the operation.

Writing

Figure 37 shows the use of the chain and multiple flags on writing. The operation is assumed to be performed on a type of unit where the length of the blocks is not fixed (e.g., magnetic tape). Note that a new block is started whenever the count goes to zero and both the multiple and chain flags are on. When the length of the blocks is fixed, as in the case of printers and card punches, the writing operation is the same as for reading.

Reading

In Figure 38, there are three blocks of data, each block containing 15 words. Cases 2 and 3 are identical as far as transfer of information is concerned, but two different methods are used to terminate the operation. In 2, the termination is under the control of the count since the multiple flag is 1 and the operation would proceed if a bigger count were specified. In 3 the termination is under block control; the count may specify more than the block contains, but a new block is not started because the multiple bit is 0. Note also that in order to read contiguous ten-word sections of the first two blocks of data into three different locations in memory, four control words are required with chaining taking place also at the end of the first block. This is illustrated by case 7.

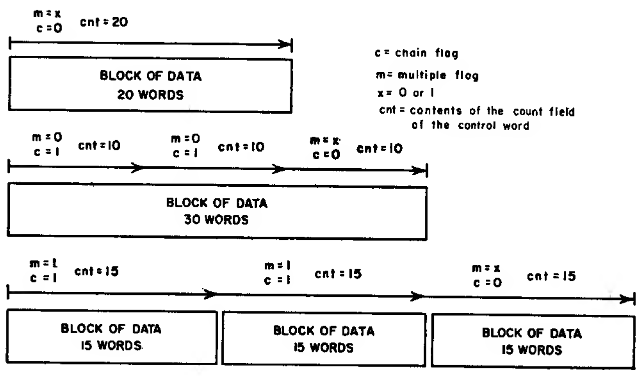


Figure 37. Writing with Chain and Multiple Flags

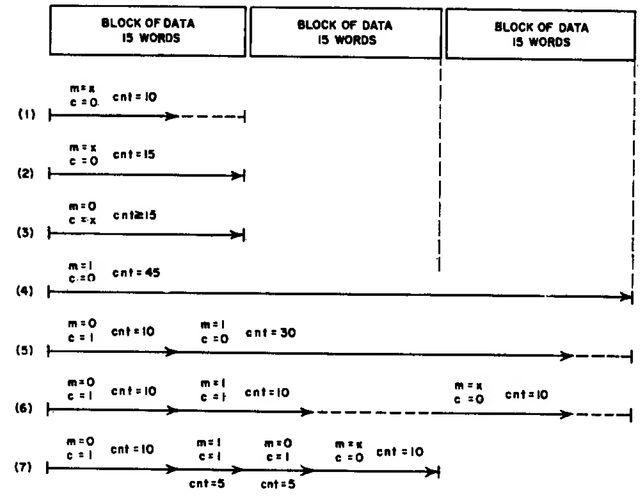

BLOCK OF DATA 15 WORDS
BLOCK OF DATA 15 WORDS
BLOCK OF DATA 15 WORDS
(1) m=x c=0 cnt=10
(2) m=x c=0 cnt=15
(3) m=0 c=x cnt≥15
(4) m=1 c=0 cnt=45
(5) m=0 c=1 cnt=10 m=1 c=0 cnt=30
(6) m=0 c=1 cnt=10 m=1 c=1 cnt=10 m=x c=0 cnt=10
(7) m=0 c=1 cnt=10 m=1 c=1 cnt=5 m=0 c=1 cnt=5 m=x c=0 cnt=10

Figure 38. Reading with Chain and Multiple Flags

Addressing of Input-Output Instructions

The basic operations involving external units are reading and writing, initiated by READ and WRITE instructions. A number of other functions are necessary for the purpose of setting up the external units for reading and writing and for monitoring these processes after they have been initiated. These tasks are accomplished by means of CONTROL, LOCATE, RELEASE, and COPY CONTROL WORD instructions. In addition to the above, there are five SEOP (suppress end of operation) instructions: READ SEOP, WRITE SEOP, CONTROL SEOP, LOCATE SEOP, and RELEASE SEOP. The SEOP instructions are identical to the corresponding instructions without the SEOP specification except that the end-of-operation indication in the control word, and therefore in the indicator register, is not turned on at the normal and successful completion of the operation. As a group, the instructions related to the external units are called input-output instructions.

The above instructions are all that are needed to control any external unit regardless of speed or type. The computer and exchange, in general, do not differentiate among units which have different characteristics. The exact interpretation of how a unit responds to each instruction is a function of the design of the external unit and will be described in connection with each unit.

External Unit Addresses

There are two levels of external unit addresses. The first level address is called *channel address*. This address is used in all input-output instructions to specify the external device or set of devices to which the instruction applies. A second level address, called *selec-*

tion address, is used for selecting one of a set of devices connected to a control unit.

Channel Address

The channel addresses correspond to the 32 physical exchange channels and are assigned the numbers 32 through 63. Within the range 32 through 63 the specific assignment of the addresses is arbitrary and depends upon the specific units in an installation. Once installed, the channel address of a specific unit (or a set of units when more than one unit is accommodated by a control unit) remains fixed. The channel address is specified by the left effective address of all input-output instructions.

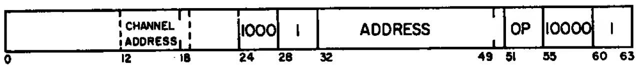
The channel address determines the priority a unit receives from the exchange in response to its service requests for bytes of data: the lower the channel address of a unit, the higher is its priority. Consequently, units which cannot interrupt their operation will normally have lower channel addresses than those having weights of zero. The same priority applies also to the transmission of words of data between the exchange and the core storage.

Selection Address

The selection address selects one unit among a set of units accommodated by a channel address. The selection addresses are assigned to a set of units independently of other such sets, usually as 0, 1, 2, 3, etc., and are specified by means of the right effective address of a LOCATE instruction. All READ, WRITE and CONTROL instructions apply to units last selected by means of a LOCATE instruction.

Format

All instructions related to external units use a full-word format, as shown below.



The effective address of the first half-word, called the left effective address, of all input-output instructions specifies the channel address. The address is seven bits long and appears in bit positions 12 through 18. The channel address register in the program interruption mechanism is also seven bits long and corresponds to bit positions 12 through 18 when addressed. The effective word address of the second half-word of the instruction, called the right effective address, contains information peculiar to the particular instruction: it can specify a control word address in the core storage (READ, WRITE, COPY CONTROL WORD), a selection address (LOCATE), control information which is decoded by the external unit (CONTROL) or it can be left blank (RELEASE). The right effective address is 18 bits long, and it appears in bit positions 32 through 49. Some instructions use only part of this field.

The addresses of both half-words of the instruction can be indexed. In the first half-word, index arithmetic extends over the full 24 bits of the channel address field of the instruction and the whole value field of the index word, including its sign, but only bits 12 through 18 of the effective address are used. The rest is ignored by the exchange. In the second half-word, index arithmetic extends over bits 32 through 50 of the instruction word and the whole value field of the index word, including its sign, but only bits 32 through 49 are used.

Input-Output Instruction Indicators

In order to permit effective simultaneous operation of the computer and several external units, all input-output operations are interlocked with the central program interruption system. This interlocking is accomplished by means of two types of indicators: the input-output reject indicators and the input-output status indicators. Indicators belonging to these two groups can be turned on only as a result of an input-output instruction or some operator interference. In addition to the above, input-output operations can activate other indicators of a more general nature. The latter indicators are described in this section only as far as they apply to input-output instructions.

General Indicators

Exchange Control Check (EK). Exchange control check is used to indicate that the exchange has failed to function properly in a manner that cannot be identified with any particular channel. It usually signals the discovery of major control and addressing errors such as inserting a byte of data into another channel's control word. The turning on of EK does not terminate the performance of the current input-output operations, if any, in the exchange.

Address Invalid (AD). The purpose of this indicator is to signal errors in the information contained in an instruction. In the case of input-output instructions, this indicator is turned on if the right effective address of a READ, WRITE or COPY CONTROL WORD is 0 through 31 or exceeds the largest address of main storage. When an error causing the AD indication is found, the processing of the instruction is terminated. The unit is never started and the contents of the channel's control word location in the exchange storage are not altered.

Programming Note

The data-store and data-fetch indicators are never turned on by an input-output operation. Address monitoring is not effective on input-output operations, and data can be read into and written from the protected core storage area as defined by the upper and lower boundary registers in conjunction with the boundary control bit. The exchange, however, cannot transfer data to or from locations 0 through 31. Similarly, all control words, including the first one specified by the input-output instruction, can be fetched from any existing core storage location, except locations 0 through 31.

Input-Output Reject Indicators

When the computer sends an input-output instruction to the exchange, it usually must wait a few microseconds until the exchange has interrogated the status of the addressed unit to determine that the instruction can be accepted. As soon as the exchange signals the acceptance of the instruction, the computer proceeds to the next instruction. If the exchange cannot accept the instruction, it responds by means of one of the three reject signals, turning on the corresponding indicator in the indicator register:

EKJ	Exchange Check Reject
UNRJ	Unit Not Ready Reject
CBJ	Channel Busy Reject

The reject indicators have a mask bit that is permanently set to one. Consequently, if the program interruption system is enabled, any of the three reject indicators will always interrupt the program. This interruption occurs before the computer has proceeded to the next instruction. It may, however, not follow immediately after the input-output instruction causing it if another indicator having a higher priority is also turned on during the execution of the input-output instruction. In such a case the higher priority indicator will interrupt first.

The reject indicators are permanent and hence, once they have been turned on, they remain on until they cause a program interruption or are explicitly set to zero by means of BRANCH ON INDICATOR. They are not reset by an input-output instruction that is accepted by the exchange.

If more than one condition exists that could reject an input-output instruction, only the reject having the higher priority is given. As an example, if an instruction is given to a channel which is not ready and is busy or waiting to make an interruption, only the UNRJ indicator is set up.

When an input-output instruction is rejected, the unit is not started, the control word for a READ or WRITE is not fetched, and the contents of the channel's control word location in the exchange storage are not altered. The following is a description of the reject indicators in the order of decreasing interruption priority.

Exchange Check Reject (EKJ). This indicator is turned on when an error is detected by the exchange in the course of testing and setting up the present instruction. It can be caused by:

1. Any malfunctioning of equipment that is detected during initiation of the operation. This includes parity errors in the portions of the instruction word sent to the exchange and errors discovered during the unit's status test.
2. Specification of channel addresses that are not available to the programmer. An unavailable channel is one that is not installed in the particular system. It does not include channels that are installed but are not operative because no units have been provided for them. The test applies only to the 7-bit channel address portion of the left effective address, bit positions 12 through 18. The remaining bits of the field are ignored.

Unit Not Ready Reject (UNRJ). This indicator is turned on when an instruction is given to a channel that is not ready to be operated. The unit-ready status bit in the control word is zero. This happens when the unit accommodated by the channel is in the not-ready status or no unit is attached to the channel.

Channel Busy Reject (CBJ). This indicator is turned on when an instruction is addressed to a channel that is still busy performing a previous instruction or which is waiting to make a program interruption.

Certain operations, such as rewinding tape, may be completed by the unit after disconnecting from the exchange. New instructions for channels performing such operations can be accepted, provided another unit is selected by means of a LOCATE instruction. In these cases CBJ is not turned on.

Input-Output Status Indicators

When an operation concerning an external unit has been terminated, the status at that time is recorded in the control word. When certain input-output conditions are satisfied, the status bits turn on the corresponding status indicators to cause program interruption. At the time of such an interruption, the channel address of the external unit is entered into the channel address register. The input-output status indicators are permanent and their mask bits are permanently set to one. The following is a list of the status indicators in the order of decreasing interruption priority.

EPGK	Exchange Program Check
UK	Unit Check
EE	End Exception
EOP	End Of Operation
CS	Channel Signal

The time at which the status bits in the control word are turned on depends upon the bits and their causes. The most common types of indications are those associated with reading or writing which are discovered by the external devices (data errors, out of material, and so on). These conditions are signalled to the exchange, a block at a time, at the end of the block in which the conditions are discovered, and, since they always terminate the operation, they cannot be on while the operation is in progress. Other conditions such as specification of an invalid core storage address can turn on the corresponding status bit before the operation at the unit has been terminated. The corresponding indicators in the indicator register, however, can be turned on and the program can be interrupted only after termination of the current operation.

If two or more units are accommodated by a single channel address (attached to a channel by means of a single control unit), the status bits can be turned on only by that unit that has been last selected by means of a LOCATE. (The channel signal is an exception; it can be turned on by any unit.)

Since the input-output status is first recorded in the control word and the corresponding indicators in the indicator register are subsequently set up only if the conditions described in "Input-Output Indicators" are satisfied, the cause and meaning of input-output status indicators will be explained by describing the control word status bits. The following is a detailed description of the status bits.

Exchange Program Check (EPGK)

This bit is turned on when an operation concerning an external unit is terminated prematurely by a programming error. Among the causes are:

1. An instruction is received that the selected unit is not designed to perform, such as a READ given to a printer.
2. The eight low-order bits of the right effective address of a CONTROL instruction contain a code that is not defined for the unit.
3. An instruction is received that the selected unit is unable to perform because of its present condition; e.g., a backspace CONTROL given to a tape unit with the tape at the load point.
4. The exchange attempts to transfer a word to or from a core storage location to which it does not have access (locations 0 through 31, or locations having addresses above the maximum available address in a particular installation). This can be caused by an invalid data word address, an invalid refill field on chaining, or by specifying in a READ or WRITE a control word address that exceeds the available core storage.

Programming errors, like invalid instructions, nonexistent control codes, or operations that are impossible because of the current status of the unit, are discovered by the external units when the units attempt to initiate the corresponding operations. The EPGK status bit in these cases is turned on a short time after the instruction is accepted by the exchange. The computer, however, will always have proceeded to the following instructions by the time the program interruption is made. The external unit to which the instruction is addressed is never started. Note that when a READ or WRITE is issued to a unit that cannot perform it, the control word for the addressed channel still is fetched.

In the case of invalid control word or data word addresses, the error is discovered at the time when the exchange attempts to fetch the word specified by the invalid address. The operation always proceeds normally until the invalid address is encountered. If the refill field of a control word specifies an address that is not accessible to the exchange, the operation proceeds until the word count in the old control word is exhausted. All data specified by the old control word are transferred normally, and the exchange at the time of program interruption contains the old control word with a count of zero. If a data word address is invalid, the operation is terminated only after data transfer up to the invalid address is completed. The control word at the time of termination refers to the core storage location following the one that is responsible for the termination.

When a READ or WRITE is terminated with EPGK because of an invalid core storage address, the unit always has been started and at least the first block of data has been moved at the unit. If the error is such that no data can be fetched for a WRITE, a word consisting of zeros is recorded at the unit. This happens when the original control word address or the first data word address is invalid. In the corresponding case during a READ, no data are transmitted to core storage. In either case, if the first control word cannot be fetched because of an invalid address, the contents of the channel's control word location in the exchange are not modified.

When more than one block is read or written, the restarting of the unit depends upon the operation and upon the relative position of the invalid address. On reading, the invalid address is discovered when the word being sent to that address has already been assembled in the exchange and, consequently, the block containing the word is always moved at the unit. On writing, the new block is not started if the invalid data word address specifies the first word of the block; otherwise, the block is started.

When the chain and multiple flags are on, the new control word applies to the next block, and, as far as the unit is concerned, the exchange starts a new operation. As long as all the data word addresses in the preceding control word are valid, the exchange starts the block controlled by the new control word. This restarting does not depend upon the validity of the refill address that specifies the control word.

The exchange-program-check status bit in the control word is turned on as soon as the exchange discovers the condition responsible for the indication, even though the computer can be interrupted only at the end of the block. These errors are always of the type that preclude a proper completion of the operation. Hence, exchange-program-check and end-of-operation bits are never on simultaneously. The turning on of the EPGK bit is not associated with any change in the unit-ready status bit.

Programming Note. When a WRITE is terminated under block control, it is possible to get an invalid address indication even if the operation is completed before the word specified by the invalid address is used. On writing, the exchange fetches new data words and control words ahead of the time when these words are actually needed in the operation. As a result, a properly completed operation is terminated by EPGK if the address of the word following the one last written is invalid. If the word last written is the last word specified by a control word in a chain of control words, EPGK is given if the refill field of the current control word is invalid and its chain flag is on or if the first data word address specified by the next control word is invalid. Note, however, that EPGK in the above cases is given only when the unit terminates a WRITE. It never occurs on reading or when a WRITE is completed under count control.

Unit Check (UK)

This bit is concerned with data errors and malfunctioning of the recording medium or equipment that can be identified with a particular channel. Among the causes of unit check are:

1. Data errors. These can be errors introduced in the information that is read or written or in control codes or selection addresses associated with CONTROL and LOCATE instructions, respectively. The following are the ways in which these errors can be discovered:
   a. The input-output unit can discover by means of parity, echo or some other type of checking an uncorrectable error in the data read or written. The error condition is signalled to the exchange at the end of the block in which it is discovered. On reading in the ECC mode, correctable error indications are suppressed.
   b. The exchange can discover an uncorrectable error in the data read or written. This can be a parity error in the byte received from the unit or an error discovered at the error checking and correction station. When an uncorrectable data error is discovered in the ECC mode, the data word is forwarded with the ECC bits unaltered.
   c. The exchange can discover a parity error in the main core storage address when fetching or storing data words. When this happens, reference to that core storage location is suppressed. On reading, the data word being sent to that address is ignored; on writing, a word consisting of all zeros is written.
   d. Some data may have been lost because exchange capacity is exceeded. This can arise when a unit's byte service or data word transfer has not been performed on time. When a byte or a word of data is missed on reading, the following data are shifted so as to fill the gap. On writing, zeros are written in place of the unavailable data.
   e. The input-output unit can discover a parity error in control codes and selection addresses associated with CONTROL and LOCATE instructions, respectively.
2. Malfunctioning of parts of the exchange affecting this channel only, and introduction of errors that prohibit the continuation of the operation. Examples of such errors are parity errors in the main core storage address when fetching a control word, any uncorrectable errors discovered in the control word after the control word has been fetched, and any parity errors discovered during the control word modification cycle.
3. Malfunctioning of the unit or the control unit. Examples of such malfunctioning are card-feed jam, broken magnetic tape, or failure of some components.

4. The operation of certain units has been interrupted by a depression of the STOP key. Most units, however, are designed so that a depression of the STOP key interrupts the operation without causing unit-check. On these units the operation is automatically resumed as soon as the ready condition is restored.

If a unit check is caused by uncorrectable data errors in a READ or WRITE, data transfer proceeds normally until the end of the current block, at which time the operation is terminated and both the UK and EOP control word status bits are turned on. Regardless of the setting of the multiple block bit, the operation never proceeds beyond the block containing the error. A data error thus can cause the EOP indication to be given before the last block specified in the instruction has been reached. The program must use the COPY CONTROL WORD instruction to determine how far the operation has progressed and whether or not all blocks have been read or written.

When a data error indication is caused by a parity failure in a control code or selection address, the decoding of the information is suppressed, and the operation is not executed. As in the case of other data errors, the UK indication in this case is accompanied by EOP, but the EPGK indication for invalid control codes is always suppressed.

When a unit check is caused by any type of malfunctioning in the unit or in the exchange, or when the operator interrupts the operation, data transfer is immediately terminated, and only the UK bit is turned on. If the new control word cannot be fetched because of an error in the core storage address of the control word, the original contents of the exchange control word location are not altered. If the fetching is successful but the control word has an uncorrectable error, the new control word in its incorrect form replaces the old one. Similarly, the exchange control word location contains an incorrect control word when the error is discovered by means of parity checking in the modification cycle. The EOP bit in the above cases is not turned on regardless of how late in execution of the instruction the error is discovered.

Whenever a READ or WRITE is terminated with a unit check for any of the reasons listed above, the unit has been started and the first block of data has been moved at the unit. This takes place even if the first control word cannot be fetched. In case of WRITE, a word consisting of zeros is written if the first data word cannot be obtained.

The state of the unit-ready bit after a UK depends upon the cause of the UK indication. If the indication is due to some malfunctioning in the unit or control unit or due to the depression of the STOP key, the unit-ready bit is off. On the other hand, malfunctioning of the exchange and errors introduced in the transmission of data, control codes, or selection addresses do not cause the not-ready status.

End Exception (EE)

This bit is used to signal a number of exceptional conditions usually associated with the recording medium or some subdivision of the data to be transmitted. It indicates that an operation has been terminated because of conditions such as:

1. A unit reached an out-of-material condition (e.g., empty card hopper, full card stacker, end of tape during writing).
2. A tape mark has been sensed on tape during reading or spacing.
3. The erase key has been pressed on the console.

When an end-exception indication is caused, data transfer is terminated at the end of the current block. At the same time the EE status bit in the control word is turned on. The ready status of the unit may or may not be affected by end-exception. If the EE indication is due to a condition which requires operator intervention (e.g., empty card hopper), it causes the unit-ready bit to be turned off. Other causes of the EE indication do not affect the unit-ready status bit (e.g., tape mark).

End of Operation (EOP)

This bit in the absence of the UK bit indicates that an operation initiated for the unit has been completed as specified by the instruction and its control words, if any. EOP in conjunction with UK indicates that an uncorrectable data error has been discovered in the last block. The EOP indication is given for all input-output operations except COPY CONTROL WORD and except when an SEOP (suppress end of operation) instruction is completed without causing a UK or EE indication.

When no uncorrectable data errors are discovered in a READ or WRITE, the EOP bit in the control word is turned on at the end of the block in which data transfer is completed. Data transfer is considered as completed when all the words as specified by the word count in conjunction with the chain and multiple flags have been read or written. When uncorrectable data errors are discovered, both the EOP and UK indications are given at the end of the block in which the error is discovered, regardless of whether or not the operation is completed. As long as the above conditions are satisfied, the turning on of the EOP bit is not affected by any exceptional conditions as indicated by the presence of the EE status bit.

In the case of CONTROL and LOCATE instructions, the EOP status bit is turned on when the specified operation is completed, except in the case of rewinding of tape, which is completed after disconnecting from the exchange. In the case of this exception, the EOP bit signals the initiation of the operation at the unit, and it follows soon after acceptance of the corresponding instruction by the exchange.

The EOP indication signalling the completion of a RELEASE instruction is given at the time when the execution of the released operation is terminated; if the unit was not operating at the time the RELEASE was given, the EOP indication comes on immediately after the control word is reset to zero.

The COPY CONTROL WORD instruction is always completed at the time the exchange allows the computer to proceed to the next instruction. No EOP indication results from the execution of this instruction.

When any of the five SEOP instructions are given, the SEOP bit in the control word is set to one during the initiation of the operation. As a result of this, the EOP bit in the control word and the indicator register is not turned on upon completion of the operation unless a UK or EE indication accompanies the EOP indication.

Channel Signal (CS)

This bit comes on when the SIGNAL key on the unit is depressed, and whenever the unit changes from not-ready to ready status. This change of status can be the result of a manual intervention or can be caused by the completion of rewinding of tape. The main purpose of channel signal is to provide a means of communication from the operator servicing the external units to the computer. The CS indication can be interpreted by means of programming as a request for READ or WRITE instructions. It is also used for initial program loading.

Channel signal is treated in the same way as other input-output status indicators, but is not necessarily related to the initiation, execution, or termination of any external unit operations. It can originate at any time regardless of whether or not the unit is operating and has been previously selected by a LOCATE instruction. If a CS is sent by a unit while the same unit or another unit connected to the same channel is engaged in data transmission, the CS status bit in the control word is turned on immediately without waiting for the end of the operation or the end of the block. This bit, however, never affects the progress of the current operation, and the corresponding indicator bit in the indicator register is turned on only after the termination of the operation.

As a result of the above, CS can be the only input-output status indicator turned on in an input-output status report, or it can come on together with any combination of the other status indicators. In the latter case the CS status bit normally is not associated with the operation whose termination is signalled by the other status indicators. When more than one external unit is connected to a common control unit, there is no way of identifying the unit which sends the channel signal. To the program, the CS appears to come from the common control unit regardless of which unit caused it.

Summary of Input-Output Status Indication

When an input-output instruction is terminated, the status bits can appear in various combinations: it is possible to have none of the above status bits on (an SEOP type of instruction), any one of the above four can be on, or a few of them can be on simultaneously.

If the EOP bit alone is on, the operation has been completed successfully as specified without detecting any errors other than those corrected by the ECC system.

The EPGK, UK, and EE indications provide for exceptions to the normal ending of the operation. Some of the exception indications, however, do not preclude an EOP indication. When a data error occurs, or an exceptional condition is discovered that does not prohibit complete execution of the instruction, the UK and EE indications can be accompanied by EOP. Examples of such cases are a byte parity error (UK and EOP) and spacing over a tape mark (EE and EOP).

EPGK always indicates a type of error that interferes with proper execution of the instruction. Hence, EPGK and EOP are never on together. The same applies to UK if the indication is caused by a condition other than a data error (e.g., a card jam). In the case of EE, EOP is absent if the condition causing the EE indication precludes proper completion of the operation, such as running out of material prematurely.

When more than one exceptional condition is discovered, the exchange turns on all the exceptional indications which apply to the individual conditions. The EOP bit, however, is never actuated when EPGK is on or a UK has been caused by conditions other than data errors. Thus, there is no distinction between data errors and equipment malfunctioning once a programming error has been discovered, as in either case EPGK is accompanied only by UK. Multiple indications can arise in cases such as when the exchange discovers an invalid data word address in the same block in which a data error occurs (EPGK and UK), the unit runs out of material (EPGK and EE), or both conditions occur together (EPGK, UK and EE). Note that when the unit runs out of material and a data error is discovered in the last block, the EOP bit accompanies UK and EE regardless of whether or not the operation is completed.

Figure 39 lists all the possible causes of termination of input-output operations and the corresponding sets of status bits that signal the termination. The channel signal indication is caused independently of the current operation and can accompany any of the sets of bits.

Input-Output Status Bits Not Represented by Indicators

Two of the control word status bits, unit-ready and suppress end of operation, are not represented by indicators in the indicator register, but they affect the turning on of other indicators reflecting the progress of input-output operations. These two bits are located in the control words in the exchange, and are always accessible to the program by means of the COPY CONTROL WORD instruction.

Unit Ready

The unit-ready bit indicates whether or not the unit currently selected on the channel is in a condition to be operated. The bit is on whenever the unit can accept an instruction from the computer or is executing an operation. When the bit is off, the unit is in the not-ready status and cannot be operated by the computer. A channel can be in the not-ready status because of the following reasons:

1. The unit accommodated by the channel cannot operate because of conditions such as out of material, operator stop, power off, control error or mechanical malfunctioning.
2. No unit is attached to the channel.

Cause of Termination				Operation Completed?	Indication Given
EPGK	UK[2]	UK[1]	EE		
				Yes	EOP, no indication for SEOP instr.
			X	Yes	EE, EOP
			X	No	EE
		X		Yes	UK, EOP
		X		No	UK, EOP
		X	X	Yes	UK, EE, EOP
		X	X	No	UK, EE, EOP
	X			No	UK
	X		X	No	UK, EE
	X	X		No	UK
	X	X	X	No	UK, EE
X				No*	EPGK
X			X	No*	EPGK, EE
X		X		No*	EPGK, UK
X		X	X	No*	EPGK, UK, EE
X	X			No*	EPGK, UK
X	X		X	No*	EPGK, UK, EE
X	X	X		No*	EPGK, UK
X	X	X	X	No*	EPGK, UK, EE

UK[1] Data errors (causes listed in paragraph 1 of "Unit Check".)
UK[2] UK due to causes other than those listed in paragraph 1 of "Unit Check"
* See Programming Note in "Exchange Program Check"

Figure 39. Termination Causes

When a unit is not ready, it remains inoperative until an operator intervenes. An exception occurs during certain operations such as rewinding of tape; the unit may be in the not-ready status while the operation takes place, but the unit-ready bit is turned on again at the completion of the operation.

The unit-ready status bit cannot directly give rise to program interruption, but it can cause other indicators to be turned on. When an instruction is given to a unit which is not ready, the UNRJ indication is sent to the computer. Once the operation is initiated, however, the exchange ignores the status of the unit-ready bit. It depends upon the design of each unit whether changing to not-ready status during an operation generates a corresponding indication. On most units the manual STOP key will merely remove the ready status at the end of the current block without forcing unit check. Thus, the operator may stop the unit between blocks and hold up a multiple-block operation until he presses the start key, without cancelling the operation. On the other hand, if the unit becomes not ready because of running out of material, it always causes the end-exception indication, while not-ready status due to any malfunctioning is always accompanied by UK. If UK or EE accompanies the change to the not-ready status, the current operation is always terminated.

Suppress End of Operation (SEOP)

The suppress-end-of-operation bit is turned on in the control word whenever an SEOP type operation is initiated at the corresponding channel. Its purpose is to suppress the turning on of the EOP status bit in the control word and the corresponding indicator in the indicator register when an SEOP instruction is completed without encountering any exceptional conditions. The SEOP bit is always turned off at the time the operation is terminated. This is the instant when in the case of successful completion the EOP bit in the exchange control word is normally turned on.

When a data error is encountered or some other exceptional condition is discovered (which in the case of the non-SEOP instructions does not preclude the EOP indication upon the completion of the operation), the EOP indication is not suppressed. In this case the EOP bit as well as the bit indicating the exceptional condition (UK or EE) are turned on, and program interruption is permitted. Similarly, an input-output status report is permitted when an SEOP instruction is terminated prematurely because of conditions such as discovery of invalid core storage addresses, equipment malfunctioning or running out of material. In these cases, the EOP bit is off, and only the bit signalling the cause of termination (EPGK, UK or EE) is turned on. When an SEOP instruction causes an input-output status report, the SEOP bit is removed at the time the status bits in the exchange control word are turned on.

Setting up of Input-Output Indicators

The status of an exchange channel at any time is described by the unit ready, exchange program check (EPGK), unit check (UK), end exception (EE), end of operation (EOP), channel signal (CS), and suppress end of operation (SEOP) status bits in the control word. As mentioned previously, EPGK, UK, EE, EOP, and CS have corresponding indicators in the indicator register. When an operation concerning an external unit has been terminated and at least one of these five status bits in the control word is on, the exchange attempts to set up the corresponding indicator or indicators and thus cause an interruption of the program. All of the status indicators for a particular channel are set up as a group at one time. At the same time, the address of the channel causing the interruption is entered into the channel address register.

The status indicators in the indicator register and the channel address in the channel address register can be set up only if both of the following conditions are satisfied.

1. The five input-output status indicators (EPGK, UK, EE, EOP, and CS) in the indicator register have previously been reset to zero either by permitting an interruption to occur or by turning them off by means of a BRANCH ON INDICATOR instruction.
2. The interruption system is currently enabled.

If either condition is not satisfied, the status bits remain set in the control word, and another attempt to turn on the indicators is made a short time later. While the status bits are in the control word, further instructions for this channel are rejected by means of a channel busy reject. If both conditions are met, the indicators and the channel address are set up, and the corresponding status bits in the control word are reset to zero. As soon as the status bits in the control word have been cleared, a new instruction for the channel can be accepted.

These two conditions permit the program to interrogate the channel address register and make any other tests without interference from other input-output status reports. Condition (1) protects the channel address register and prevents any new input-output status indicators from being set while at least one status indicator due to a previous status report is on. Condition (2) prevents the destruction of the contents of the channel address register after the last indicator belonging to a given set is reset, but before the subroutine whose execution is caused by the indicator has had time to interrogate the channel address register.

Since the input-output status reports cannot be made while the interruption system is disabled, it is not possible to turn the input-output status indicators on without immediately getting an interruption from the highest priority indicator. If, after getting this interruption, successive interruptions due to the same status report are not desired, the program can test and reset the remaining indicators of the set. In any case, regardless of how status indicators are turned off, the enabling of the interruption system after resetting of the last indicator is used as an indication that all the action associated with the status report of a particular channel has been taken, and that a new set of input-output status indicators can be turned on. The system, therefore, should not be enabled until all information associated with the current input-output status report has been acted upon or stored elsewhere for later use.

It follows from the above two conditions that only one channel at a time can cause input-output status interruptions. While interruptions due to one channel are being processed, the exchange holds up any other status reports due to the same or other channels. When the computer has cleared the current set of status indicators and the interruption system is enabled, the exchange presents the next set of status indicators as if they had just occurred.

When, as a result of operating in the disabled mode, more than one channel is waiting to make a status report, the order of interruption is determined by a scanner in the exchange. The channel that gets the highest priority is the one specified by the current position of the scanner. This order is not related to the sequence in which the corresponding operations were terminated, and normally is random. However, when an input-output instruction is issued, the scanner is set to the channel to which the instruction is addressed. Consequently, if, while still disabled, the program issues a COPY CONTROL WORD immediately followed by BRANCH ENABLED, the highest interruption priority is assigned to the channel to which the COPY CONTROL WORD is addressed. If this channel is not ready to present its status report, the subsequent priorities are assigned in the order of ascending channel addresses, looping around to the lowest address once the highest address of the installation is reached. This synchronism is lost when one channel has presented its status report, since the scanner may continue to scan without being able to interrupt.

As soon as the interruption system has been enabled, the indicator register is in a position to accept a new input-output status report. The turning on of the next set of input-output status indicators, however, may not follow immediately after the enabling even if the exchange does have a channel that is ready to interrupt. The exchange assigns to the input-output status reports a lower priority than to such activities as chaining and transmitting data words to or from core storage. Since each of the latter functions occupies a 10-microsecond cycle, there can be a delay of a few such cycles in turning on the input-output status indicators.

No mask bits are available to prevent status indicator interruptions, as there are for some of the other indicators. If a program interruption due to input-output status is not desired, the system can stay disabled, and the status bits can be obtained from the exchange by means of COPY CONTROL WORD. Thereafter, the status bits in the control word can be reset by means of RELEASE SEOP, thus clearing the channel for the next input-output operation. The EOP indication after a normal completion of the operation can be avoided by using the SEOP instructions in lieu of the normal set of instructions.

Channel Address Register

The channel address register identifies the channel responsible for the current input-output status indicators. Whenever input-output status indicators are set up, the channel address of the unit is placed in the channel address register. The register has word address 5 and corresponds to bit positions 12 through 18. These bit positions are the same as those of the channel address in an input-output instruction.

The channel address of a unit is retained in the register until the system is in a condition to accept the next set of input-output indicators, as described in the preceding section. At this time, the channel address is reset to zero. Thereafter, at any time when the interruption system is enabled, the exchange may make a new status report and load a new address in the channel address register. If the register is addressed at an instant when its contents are changed, the instruction-check or instruction-reject indicator is turned on, depending upon the instruction involved. For this reason the channel address registers should be addressed only when handling interruptions.

The channel address register can be read out only. The bits are lost whenever information is loaded into it. In this sense the effect of the register is the same as that of location 0.

Instructions

This section describes all input-output instructions. It also lists all the input-output reject and status indicators that may be turned on by each instruction.

Read (RD)

This instruction initiates a reading operation for the external device specified by the channel address. If more than one device is accommodated by a channel address, the instruction applies to the device last selected by a LOCATE instruction.

The right effective address specifies the first control word to be used. The control word, in turn, supplies the information defining the data addresses in core storage and the number of words to be read from the external device. An EOP (end of operation) indication is given when the data transfer is completed as specified. The basic reading operation can be modified by means of the chain, multiple and skip flags.

Indicators

All input-output reject and status indicators, except channel signal.

Write (W)

This instruction initiates a writing operation for the external device specified by the channel address. It is analogous to the READ, except that the skip flag in the control word is ignored, and uses the same indicators.

Programming Note

The exchange does not obtain the control word for a READ or WRITE from core storage until after the computer has proceeded to the next instruction. Additional control words in a chain are obtained only when the operation has progressed to the point where each control word is needed. The programmer must therefore exercise caution to avoid altering control words in core storage that are being used in current operations, unless he specifically desires to make alterations in flight. Normally, control words are set up before an operation starts and remain unchanged until the end is signalled.

It must also be realized that some kinds of data errors are not detectable until the end of the block is reached. It is easier and safer not to use portions of a block until the whole block has been read into core storage, nor to alter the core storage locations of portions of a block being written until it is certain that the entire block has been written correctly.

Control (CTL)

This instruction is used to initiate the performance of certain functions at the external devices. Among these functions are:

Turning on the RESERVED light
Sounding the gong
Rewinding of tape
Backspacing of tape
Writing of tape mark

The right effective address contains control information that is decoded at the device specified by the channel address. If more than one device is accommodated by a channel address, the instruction may apply to the control unit or the device last selected by a LOCATE instruction, depending upon the control function. The code specifying the control function appears in the rightmost byte of the address, bits 42 through 49. Only this byte is sent to the unit; the rest of the right effective address is ignored. An EOP indication is given when operation is completed as specified. No distinction is made between instructions that cause some operations to be performed and those that are superfluous. The latter case can arise when the instruction specifies a state or a mode of operation that is already set up in the addressed unit.

An exception to the normal process of the execution of an instruction takes place in the case of rewinding of tape. In order that another tape unit connected to the same control unit may be used while rewinding takes place, the EOP indication is given and the operation at the exchange is regarded as completed immediately after the initiation of the operation. The unit subsequently proceeds with rewinding and is in the not-ready status. When rewinding is completed, a channel signal is given and the unit automatically assumes the ready status. After the EOP indication following the initiation of the instruction is received, another tape unit can be selected by means of a LOCATE instruction.

INDICATORS

All input-output reject and status indicators.

PROGRAMMING NOTE

When a parity error is discovered on transmitting the control code from the exchange to the unit, the execution of the CONTROL instruction is suppressed. The UK indication in this case is always accompanied by EOP, but an EPGK indication due to an invalid code is suppressed.

Locate (LOC)

This instruction is used to select one of several devices accommodated by a single channel address. Once a device is so selected, all operations giving this channel address refer to the same device until a LOCATE is given to select another device on that control unit.

The right effective address of the instruction contains the selection address for the channel specified by the left effective address. The selection address is three bits long and appears in bit positions 47 through 49 of the instruction. It is sent to and decoded at the addressed control unit. The remainder of the right effective address is ignored. An EOP indication is given when the operation is completed as specified.

In the case of the LOCATE, an exception to the regular instruction rejection occurs. Since it may be desired to switch from a device that is in the not-ready status to one that is ready, the LOCATE is accepted regardless of the state of the unit-ready status bit in the control word. Thus, a Unit Not Ready Reject will not be given for a LOCATE when the device originally selected is not ready, or the new device addressed by the right effective address of the instruction is not ready. Normally, the common control unit will be ready and will terminate the instruction. The control word in exchange storage must be manually cleared from the exchange maintenance console.

INDICATORS

Exchange check reject (EKJ), channel busy reject (CBJ), exchange program check (EPGK), unit check (UK), and end of operation (EOP).

PROGRAMMING NOTE

When a parity error is discovered on transmitting the selection address from the exchange to the unit, the execution of the LOCATE instruction is suppressed, but an EOP indication always accompanies the UK indication.

Release (REL)

This instruction immediately terminates any operation in progress at the external unit specified by the channel address and resets to zero the EPGK, UK, EE, EOP, CS and SEOP status bits in the control word. When the activities at the unit due to the released operation have ceased, the EOP indication is given.

If the addressed channel is reading or writing, data transfer is terminated at the completion of the current word. The unit then proceeds to the end of the block, whereupon all control word status bits except unit ready are reset to zero, and the EOP status bit is turned on. If a CONTROL or LOCATE is in progress at the specified unit, the released operation is executed normally except that at the completion of the operation all control word status bits indicating special conditions are suppressed and only the EOP bit is turned on. If no operation is being executed by the unit at the time the RELEASE instruction is given, all previously set status bits, except unit ready, are reset to zero and the EOP bit is turned on immediately. The COPY CONTROL WORD cannot be released since it is executed before the computer proceeds to the next instruction. The address of the second half-word of the instruction is not used and is ignored.

The RELEASE instruction is accepted regardless of the status of the unit-ready bit in the control word of the addressed channel. If there is no operation currently in progress at the unit, the status of the unit does not affect the execution of the instruction. The pertinent status bits in the control word are reset to zero, and the EOP indication is immediately turned on. If, however, a LOCATE has been issued to a channel with an inoperative control unit, or an operation has been interrupted because of the not-ready status without cancelling it, the RELEASE cannot free the channel. The RELEASE instruction is executed immediately. In order to make the channel available, the control word in exchange storage must be manually reset. The RELEASE instruction is not subject to channel busy reject. Thus, when the exchange fails to complete a RELEASE instruction because of an I-O unit malfunction, the I-O unit should be made not ready and a RELEASE instruction should be given for the unit.

INDICATORS

Exchange check reject (EKJ) and end of operation (EOP).

PROGRAMMING NOTES

It is possible for an exchange channel to turn a set of input-output status indicators on while the CPU is issuing a RELEASE instruction to that channel. As a result, the status bits due to the released operation may cause a program interruption between the execution of the RELEASE and the instruction following RELEASE. Thereafter, as soon as this set of status indicators has been cleared, the same channel will set the EOP indicator signalling the completion of RELEASE and cause another program interruption.

If a RELEASE immediately follows a READ, no data transfer takes place, but a block of data is moved at the unit. If RELEASE immediately follows WRITE, a block consisting of one data word is always written before the operation is terminated. In cases where RELEASE is given immediately after a CONTROL or LOCATE, the execution of the subject instruction is not affected except for the suppression of status indications due to exceptional conditions.

Note that in the case of manual input devices, a READ cannot be released until the operator completes the current word. If the RELEASE is issued after a word has been completed but before a new word has been started, the operator has to provide another input word in order for RELEASE to terminate the operation.

Read SEOP (RD SEOP)
Write SEOP (W SEOP)
Control SEOP (CTL SEOP)
Locate SEOP (LOC SEOP)
Release SEOP (REL SEOP)

The first four of these five SEOP (suppress end of operation) instructions are similar to READ, WRITE, CONTROL, and LOCATE, respectively, except for the use of the end-of-operation indication. At the time an SEOP instruction is given, the SEOP bit in the control word is set to one. As the result of this, the EOP bit in the control word (and therefore in the indicator register) is not turned on unless it is accompanied by unit check or end exception.

The five SEOP instructions provide a means of suppressing program interruption upon the completion of an operation when no exceptional conditions are encountered during the execution of the instruction. When an exceptional condition is discovered, as indicated by the presence of EPGK, UK or EE status bits, the interruption of the program is not suppressed. In this case, the status bit indicating the exceptional condition, as well as the EOP bit if the operation has been completed, are turned on, and program interruption is permitted.

Receipt of a channel signal during the execution of an SEOP instruction does not cause the EOP bit to be turned on. When the operation is completed, the CS indicator is set, but this does not have any effect on the turning on of the indicators pertaining to the SEOP operation.

The RELEASE SEOP instruction never causes any interruptions. It performs all the functions of RELEASE except that it does not cause an EOP indication. Thus, RELEASE SEOP can be used to turn off EPGK, UK, EE, CS as well as any previously set EOP bits in the control word. If the addressed unit is executing an instruction at the time RELEASE SEOP is given, the SEOP bit is temporarily set in the control word.

INDICATORS

Indicators are the same as in the corresponding non-SEOP type instructions, except that EOP is never turned on after execution of RELEASE SEOP.

PROGRAMMING NOTE

The EOP indication following a RELEASE always applies to the RELEASE and not to the instruction being released. Thus, a RELEASE is always terminated by an EOP indication regardless of whether the instruction released is READ, WRITE, CONTROL, or LOCATE or the corresponding SEOP instruction. On the other hand, no EOP indication is ever given at the completion of a RELEASE SEOP.

Copy Control Word (CCW)

The COPY CONTROL WORD instruction causes the current control word corresponding to the addressed channel to be sent to the core storage location specified by the right effective address. The COPY CONTROL WORD is completed before the computer proceeds to the next instruction, and no EOP indication is given.

The format of the control word as copied from the exchange is the same as that described in "Control Word Format." The status bits, bits 18 through 24, reflect the current status of the input-output unit last selected. The data word address and count fields also contain the current values and thus reflect the progress of the operation. The refill address and the three flag bits are never modified and are the same as those originally specified in the control word in core storage.

The COPY CONTROL WORD instruction is accepted regardless of the status of the unit and can be executed while an operation is in progress at the addressed channel without disturbing the operation. Consequently, it is not subject to UNRJ and CBJ. The computer program may thus monitor the progress of an input-output operation, if desired for special procedures.

Since COPY CONTROL WORD by definition does not affect the contents of the control word of the addressed channel, it cannot cause any status bits or status indicators to be turned on. As a result, all errors that preclude the completion of the COPY CONTROL WORD are signalled by turning on the EKJ (exchange check reject) indicator. In addition to the regular errors responsible for EKJ, this includes any errors discovered in the process of transmitting the control word to core storage. When COPY CONTROL WORD is terminated by means of EKJ, the control word is not transmitted and the addressed core storage location is not altered.

Caution must be exercised in using the data word address and count information obtained from the exchange if COPY CONTROL WORD is given while another operation is still in progress. During both reading and writing, the data word address of the control word specifies the core storage address to which the next exchange reference will be made. On reading, the address specified by the control word is that of the data word just being processed by the external unit. During writing, the control word always gives an address which is two words ahead of the data word just being written.

INDICATORS

Exchange Check Reject (EKJ).

PROGRAMMING NOTES

If a control word is to be interpreted as an instruction after it is brought from the exchange by COPY CONTROL WORD, it cannot be located in either of the two core storage words that are fetched by the instruction fetching mechanism immediately following the fetching of the COPY CONTROL WORD. This restriction is caused by the fact that at any time the instruction fetching mechanism may be two full core storage words ahead of the instruction being executed. This mechanism is interlocked to detect any changes in these words caused by the central processing unit, as when an instruction is modified arithmetically. When this happens, the word is refetched. The mechanism cannot, however, detect changes in these words caused by the exchange, as in the case of COPY CONTROL WORD. The above restriction, therefore, does not apply to ordinary modification of immediately succeeding instructions by the central processing unit. There are no restrictions on the use of the copied control word when it is used as the addressed operand rather than as an instruction.

COPY CONTROL WORD has a number of applications in connection with input-output operations. The instruction can be used for checking the progress of reading operations to determine that a certain block of information has been read before its processing is initiated. It also can be used to interrogate input-output status bits in the control word if no interruptions are desired.

Another application of COPY CONTROL WORD is in the case of termination of input-output operations due to exceptional causes. If an exchange program check, end exception, or a unit check occurs, the unit stops immediately after the block in which the condition is discovered. A COPY CONTROL WORD is required to determine which of the blocks in the sequence was the last one to be read or written. For example, it is frequently not possible to insure that the number of cards in the hopper of a card reader is precisely a multiple of the number of cards to be read with one instruction. An end-exception indication may occur at any one block if the unit runs out of cards. If more cards are to be loaded and the group of blocks is to be completed, the program must, with the aid of COPY CONTROL WORD, compute the remaining number of words and addresses.

Summary of Exchange Response to Instructions

The instructions listed in Figure 40 include also the corresponding SEOP instructions.

	Read Write Control	Locate	Release	Copy Control Word
1. Reject indicators available:				
Exchange Check Reject	Yes	Yes	Yes	Yes
Unit Not Ready Reject	Yes	No	No	No
Channel Busy Reject	Yes	Yes	No	No
2. Computer waits until end of operation	No	No	No	Yes
3. Status indicators available at the end of operation:				
Exchange Program Check	Yes	Yes	No	No
Unit Check	Yes	Yes	No	No
End Exception	Yes	No	No	No
End of Operation	Yes	Yes	Yes*	No
4. Channel Signal**	Available at any time except when channel is in operation			

*The End Of Operation is never turned on in a RELEASE SEOP instruction.

**A Channel Signal indication can be caused by the program only upon the completion of tape rewinding (initiated by a CONTROL instruction).

Figure 40. Exchange Instruction Response

Handling of Control Word

Zero Initial Word Count

If the count field in the control word is zero initially, it is interpreted as 2^{18}. The field is not tested for zero until the first time the control word is modified; the count at that time is $2^{18} - 1$, and counting will continue. The count field cannot specify less than one word. This treatment of the count field of a control word is compatible with the treatment of the count field of an index word.

An initial word count of zero sometimes can be a convenient way of indicating that the amount of data transfer is to be governed entirely by the external unit. Data transfer, however, is terminated by the exchange whenever the data word address exceeds the largest available address of core storage. It is therefore not possible to loop to address 0.

Use of Control Word for Indexing

While in an index register, a control word may be used for indexing like any other index quantity. A common use is to treat the data word address of a control word as a base address modifying a relative address. Indexing instructions may be used to operate on or with the data word address, the count, or the refill address.

Bits 0 to 17 form the word address, both for indexing and for use in the exchange. Bits 18 to 24 form a bit address and sign for the purpose of indexing, but they are ignored by the exchange when it obtains a control word from core storage; normally bits 18 to 24 will be zeros. Bit 25, the chain flag, may set the index-flag indicator in the indicator register during indexing operations for use by the program as needed. Bits 26 and 27 are not used for indexing operations, but they are carried along unchanged when loading, storing, or refilling index words.

The control word and index word formats are compared in Figure 35.

State of Control Word in the Exchange

The control word in the exchange always reflects the current status of a data transfer operation. When the operation is terminated, the contents of the control word, except for its status bits, are not changed until the exchange fetches the control word for the next READ or WRITE that is addressed to the same channel. It is not changed if the following READ or WRITE is rejected or is terminated because the initial control word address is not accessible to the exchange. By using COPY CONTROL WORD, the program thus can always inspect the control word and determine at what point the last operation has been terminated, or how far the current operation has progressed.

If such a reference is to be made, the data word address field in the control word always contains the address of the core storage location to which the next data word reference will be made. This field thus always reflects the current status of data transfer between the exchange and core storage. The count, on the other hand, always indicates how many words remain to be transferred between the exchange and the external unit. When any data transfer operation is terminated because of exhausting the control word count, the count is zero, and the number of words transmitted between the exchange and core storage is always the same as the number of words transmitted between the exchange and the unit. If this is the intended type of termination, it is generally not necessary to use the information left in the control word. Interrogation of the control word, however, may be necessary if the operation is terminated under block control or because of some type of error.

During a READ operation the data word address in the control word is always one word ahead of the address of the last word sent to core storage. The last word sent to core storage is the one whose reading has just been completed by the unit, regardless of whether the whole word has been completed or just a few bytes have been read. In the latter case the rest of the word is filled up with zeros. The data word address indicated by the control word thus is always also one word ahead of that of the word whose handling has just been completed by the exchange and the unit and, if an operation is currently in progress, specifies the location of the word that is currently being processed by the unit.

The count in the above case is decreased from the initial count by the number of words received from the unit. Since the exchange executes a COPY CONTROL WORD only after all pending word transfers to or from core storage have been completed, this number is always the same as the number of words transmitted to core storage, and the current count is equal to the number of words remaining to be read. If the area described by the control word has been fully used, the count is zero. The sum of the current values of the data word address and count fields is always equal to the sum of the original values of these fields. When the operation is terminated within a word, no information is available as to how many bytes of the last word have actually been read and how many have been provided by the exchange.

For WRITE, the data word address in the control word is always one word ahead of the location from which the last data word was fetched. Normally, however, the word last fetched is not the one currently being processed by the unit. In order to insure continuous data transmission, each exchange channel provides a buffer for storing two data words for the current operation. On writing, one of these buffer positions contains the word currently being disassembled while the other position in the meantime is loaded with the next word to be sent to the unit. When a WRITE is terminated by the unit, each of these buffer positions contains a complete new data word. The exchange always replenishes the buffer position last used, even if only part of the word contained in it has been sent to the unit. Consequently, since the control word always refers to the core storage location from which the next data word will be fetched, the data word address is three addresses ahead of the last word handled by the unit. The program cannot determine if the whole word or only a few bytes of the last word have been sent to the unit.

When a COPY CONTROL WORD is issued to a channel executing a WRITE, one of the buffer positions normally is in the process of being disassembled. Consequently, the data word address in this case is two words ahead of the word currently being handled by the unit. The count on writing indicates the number of words transferred between the exchange and the unit. It includes also any words whose handling has been interrupted and which have not been completely recorded at the unit. Except for the initial loading of the two buffer positions during the initiation of an operation, the count field is stepped down whenever the data word address is stepped up. Thus, unless the initial count is one, the sum of the current data word address and count is always two greater than the sum of the originally specified contents of these two fields. This relation holds also whenever the control word is copied during the execution of a WRITE.

When all data words specified by the control word have been fetched, further core storage references are suppressed. Writing, however, still proceeds until the last two words contained in the exchange buffer have been written. As each of these words is sent to the unit, the count is stepped down until it finally reaches zero. At the same time the data word address is stepped up, even though it is not used. When the operation is completed, the count is zero and the data word address refers to a location that is three words ahead of the one from which the last word was fetched. The unused data word addresses are not tested for invalidity and hence cannot interfere with a proper termination of the operation.

The following rules summarize the relation between the data word specified by the control word and that last handled by the unit when an operation is terminated.

OPERATION	CONTENTS OF CONTROL WORD	
TERMINATED	DATA WORD ADDRESS	COUNT
READ	A_n	$C_i - N$
WRITE	$A_n + 2$	$C_i - N$

A_n = address of next data word to be handled by an external unit
= 1 + address of last word read or written
$= A_i + N$

A_i = initial data word address

N = number of words transmitted between the exchange and the unit

C_i = initial count

If a reading or writing operation is terminated by a RELEASE, these rules do not hold to the extent that the last word read by the unit may not be transferred to core storage and, on writing, both of the buffer registers may not be filled up.

When an inaccessible core storage location for data transmission is specified, the error is discovered at the time the exchange attempts to transmit data to or from that location. This takes place after the control word has been modified, and hence the data word address at this time specifies a location that is one word beyond the invalid location. The status of the unit and the contents of the control word for the latter case are described in more detail in "Exchange Program Check."

When chaining is specified, the exchange obtains the new control word immediately before referring to the first core storage location specified by the new control word. On reading, the new control word is fetched when the data transfer specified by the preceding control word has been completed and the exchange has already assembled the first data word to be sent to the area defined by the new control word. This timing does not depend upon the original word count of the control word nor on the state of its multiple flag.

On single block writing, chaining normally takes place when the count in the control word is stepped from 2 to 1. When a control word on chained writing originally has a word count of one, the next control word is obtained one word time (the time required to write one word) after fetching the control word with the count of one. If the first control word of a WRITE has a word count of one, and the chain flag is on, the next control word is fetched immediately after fetching the data word for the first control word.

As a result of the buffering action of the exchange, the new control word is fetched and stored in the exchange ahead of the time it actually starts controlling the operation of the unit. The restarting of the unit at the end of a block depends upon the multiple flag of the control word controlling the transfer of the last data word of the block. The next control word takes over the control only after the unit has requested the first byte of the data word belonging to that control word. In order to accomplish this, the exchange always stores the multiple flags of the pertinent control words.

An exception to the above rules for chained writing occurs when the current control word has both the multiple and the chain flags on. In this case the new control word starts a new operation as far as the unit is concerned, and chaining takes place only after the exchange buffer has been cleared and the preceding block has been terminated. If an error is discovered in the completed block, the operation is terminated at the end of the block, and the exchange contains the old control word at the time of program interruption.

Initial Program Loading

In order that a program can be loaded into the machine without a programmed READ instruction, an initial program loading technique is provided. The initial program can be loaded from any unit accommodated by the exchange and is based on a special interpretation of channel signal. The loading is initiated by means of the INITIAL PROGRAM LOAD key, which sets up the exchange and the computer in a special mode. Once the INITIAL PROGRAM LOAD key is depressed, the first channel signal interruption causes the channel responsible for the interruption to read in a specified number of words into a specified core storage area.

The INITIAL PROGRAM LOAD key is physically located on the operator's console, although not an integral logical part of it. If the system does not contain an operator's console, it is provided in a separate box, which may be placed at a convenient operating point.

The initial program, as recorded on the medium from which it is to be loaded, must start with a control word that specifies the number of words to be read and the core storage address at which the first word of the remainder of the program is to be stored. This control word is immediately followed by the program itself. When the program has been read, the computer automatically starts execution of the new program. Specifically, the depression of the INITIAL PROGRAM LOAD key has the following effect:

1. The execution of any program by the computer is terminated.
2. The interruption system is temporarily disabled.
3. All input-output operations on the exchange are terminated.
4. All control words in the exchange are reset to zero.
5. The exchange is set to interpret the next channel signal in a special way.
6. All input-output units are reset to "initial power-on" status.
   a. If an input-output unit was in ready status before the IPL, it remains in that status.
   b. If the card punch was filling its buffer at the time of the IPL, zeros are inserted for the remainder of the unit record and the card is punched. If the punch was punching, the operation is completed. An error sensed during this process may cause an indicator to be set during execution of the program loaded by the IPL.
   c. If the card reader is in ready status when an IPL is given, a maximum of two old records may be obtained if the new program addresses the reader. Each of these records may cause the setting of indicators during execution of the program loaded by the IPL. This can be prevented by manually unloading the reader.

The depression of the INITIAL PROGRAM LOAD key performs these functions even if the computer has hung up within an operation.

The disabling of the interruption system prevents any extraneous interruptions, such as the elapsed time clock, from interfering with the initial loading until the program has set up suitable procedures for coping with such interruptions. The initial program should reset the indicator register before enabling the interruption system.

The resetting of input-output units to the "initial power-on" status has the following effect:

1. Where more than one input-output unit is connected to a common control unit, the control unit is reset to select the unit with the selection address 0.
2. Where applicable, the unit is reset to the no-ECC and odd-parity mode of operation.

As a result of the above, a program can be loaded by the initial loading technique only from a unit having the selection address 0. However, since the exchange does not differentiate between the channel signals sent by different units connected to the same control unit, the initial program loading is started by a channel upon the depression of the SIGNAL or START key on any unit accommodated by that channel address.

The first channel signal indication received after the depression of the INITIAL PROGRAM LOAD key from a unit accommodated by the exchange initiates the following sequence of events:

1. The exchange automatically stores in the control word location for the channel responsible for the channel signal a fixed control word with the following contents:

Data word address	= 4
Chain flag	= 1
Multiple flag	= 0
Skip flag	= 0
Count	= 1
Refill address	= 4

2. The exchange initiates a READ for the indicated channel, using the fixed control word.

3. The first word read from the unit is stored in the special core storage location referred to by means of address 4. Since the fixed control word is then exhausted, its refill address causes the next control word to be fetched from location 4. Location 4 at this time should contain another control word specifying the address at which the word immediately following should be stored, and the number of words to be read from the unit. Reading then proceeds normally.

4. If and when the READ operation is completed as specified by the control word or words without causing the exchange-program-check or unit-check indications, the exchange sends an initial start signal to the computer. The encountering of an end-exception condition during the reading does not affect a successful completion of initial loading as long as all the conditions are satisfied that would normally cause EOP to be given. The EOP as well as the EE indications, however, are not turned on upon the completion of the operation. The sending of the initial-start signal to the computer completes the initial program loading cycle in the exchange.

5. The computer, upon receipt of the initial-start signal, fetches and executes the instruction in the location specified by the data word address field of the control word in core storage location 4. The address of this instruction, however, is not placed in the instruction counter. Unless specified otherwise by this instruction, the interruption system after the completion of this instruction returns to the state it was in at the time the INITIAL PROGRAM LOAD key was depressed. This completes the initial program loading sequence.

The second instruction executed after the initial program loading is the one specified by the instruction counter at the time the first instruction is completed. If the first instruction did not alter the contents of the instruction counter by a branching instruction, the instruction counter at this time refers to the instruction following the one which was interrupted by the depression of the INITIAL PROGRAM LOAD key. Normally this instruction, however, cannot be properly executed. When the INITIAL PROGRAM LOAD key is depressed, the execution of the current instruction is immediately terminated. Similarly, the system terminates the interpretation and preparation for execution of the succeeding instructions. In doing this, the storage locations and registers that are altered by the execution of the current instruction and the preparation of the following instructions are not restored to the former or any other predetermined state. Therefore, it is not possible to continue the program.

As the result, the first instruction of a program loaded by means of the initial program loading technique must be BRANCH DISABLED. This instruction replaces the contents of the instruction counter and insures that the system does not return to the enabled state. If the BRANCH DISABLED is preceded by STORE INSTRUCTION COUNTER IF, the original contents of the instruction counter can be made available to the programmer. This may be useful in diagnosing programs in cases where the initial program loading technique must be resorted to because of a failure of some program previously loaded into the computer.

If the READ as described in 3 and 4 is not successful, the input-output unit comes to a stop. In order to restart initial program loading, the INITIAL PROGRAM LOAD key has to be pressed again. The next channel signal then repeats the listed sequence of events, starting with step 1.

The first channel signal received at the exchange following the depression of the INITIAL PROGRAM LOAD key starts initial program loading, but it does not turn on the CS status bit in the control word. All subsequent channel signals are interpreted in the normal way. If a channel signal is received by the exchange during the initial program loading, the CS status bit is turned on and the computer is interrupted as soon as the system is enabled. The exchange, however, cannot have the EPGK, UK, EE, or EOP status bits in any exchange channel at the time initial loading is completed.

The termination of input-output operations and the resetting of control words are equivalent to issuing RELEASE instructions to all exchange channels, except that the whole control word instead of only its status bits is reset. When the INITIAL PROGRAM LOAD key is depressed, all data transfer between external units and core storage is immediately stopped, but each unit still continues operating until the end of the block is reached. In case of tape rewinding, a channel signal is given at the completion of the operation. Note that if this is the first channel signal to be received following the depression of the INITIAL PROGRAM LOAD key, initial program loading will be initiated from the corresponding channel.

Handling of Data Errors

Error Checking

All input-output operations are extensively checked for errors introduced in the units, in the exchange, or in the communications between the exchange and its units and core storage.

Whenever data are transmitted between the exchange and the units, a parity bit is added to each byte to verify the transmission. A parity bit is also used to check the transmission of core storage addresses when transmitting data words and control words to and from core storage. Similarly, a parity bit is used to verify the execution of a number of control functions within the exchange. In the input-output units, checking is applied to the reading and recording of information. The nature of this checking depends upon the unit. It may be based on echoes, the use of a parity bit, reading at two stations, or reading immediately after recording.

As mentioned previously, whenever a data error is discovered, unit-check and end-of-operation indications are given, and the operation is terminated at the end of the current block. When such an error is discovered on writing, the error always appears at the unit to the extent that the recorded information disagrees with that sent to the unit or will cause an error indication when later read. On reading, however, a data error responsible for unit check does not necessarily have to appear in the information transmitted to core storage.

As far as checking and handling of data are concerned, there can be three stages in the execution of a READ instruction:

1. Normal reading. In this case data are transmitted from the unit to the exchange and from there to core storage.
2. Reading with the skip flag on. In this case data are transmitted to the exchange, but the sending of the assembled words to core storage is suppressed.
3. Skipping over the remainder of a block after exhausting the control word count (no chaining within block). In this case data are read by the unit, but not sent to the exchange.

The skip flag affects only the operation of the exchange. The unit is not aware of the presence of the flag, and the discovery of any data errors by the unit always causes a unit-check indication to be sent to the exchange at the end of the block. Since the skip flag may be changed at any time within a block, the state of the flag at the end of the block does not describe the operation at any previous time. In order to insure that no error indications are lost, the exchange always forwards all unit-check indications to the computer.

When the unit is skipping to the end of the block after receiving a disconnect signal (because of exhausting the control word count), the checking for data errors depends upon the unit. In the case of the card reader, for instance, data errors can be identified with a particular byte, and data error indications are suppressed when the data are not actually forwarded to the exchange. In the case of the 729 tape units this is not true. Because of the method of parity checking that is used, it is not always possible to identify an error indication with a particular byte, and hence no data error indications are suppressed.

As a result, in either of the above two cases, the program can get a unit-check indication without actually having an error in the information transmitted to core storage. In the case of some units, such as the 729's, the program does not have any means of identifying a unit check with a particular word or byte, and it cannot find out whether the error was discovered in the portion of the block sent to core storage or in the portion whose reading has been suppressed.

Error Correction

In addition to the error checking described in the preceding section, an automatic error correction process is applied to all information transferred from core storage to the exchange and, under the programmer's control, to data read from external units. This process is applied on a word basis, and permits correction of single errors and detection of double errors per word.

The error correction is accomplished by means of a set of ECC (error checking and correction) bits which are carried with the data word. Whenever a single error occurs, it is automatically corrected, and the data word is transferred without any indication to the program. A double error cannot be corrected, and when one is encountered, the operation is terminated at the end of the current block.

With regard to automatic error correction of information transferred between the exchange and the external units, two classes of units may be distinguished. Some units by their nature do not permit automatic error correction, particularly manual input units and printers. Information is sent to these units always without error correction bits, although parity bits are used to check the transmission. For input, the error correction bits needed in storage are supplied automatically by the exchange.

On other units, such as tape units, card readers and card punches, error correction may be applied at the programmer's option. These are units which may have to provide communication with other equipment not designed to handle error correction bits, but which may on other occasions provide external storage of information that remains in the system. These units are equipped with an ECC mode of operation, set by a CONTROL instruction. When the ECC mode is off, error correction bits are not recorded, nor are ECC bits expected on reading. A word in this case consists of the 64 data bits. With the ECC mode on, ECC bits are written, and error correction is applied to the information while reading. In this mode a word recorded by the input-output unit consists of 72 bits: the 64 data bits preceded by eight ECC bits.

While the unit is reading in the ECC mode, correctable data errors detected by the checking circuits at the unit are not passed on as unit-check indications to the exchange. This permits the error checking and correction station in the exchange to correct these errors. When uncorrectable data errors are encountered, either the exchange or the unit provides the error indication.

When the exchange encounters a double error, the bits of the word, including the ECC bits, are not altered. If this occurs on writing, the word sent to the unit, including its ECC bits, if any, is in error. If a double error is encountered on reading in the ECC mode, the unaltered word is sent to core storage. Consequently, another error indication is given when this word is subsequently read out: an instruction-reject or an instruction-check indication if the word is referred to by the central processing unit, or another unit-check if the word is subsequently written.

When the unit is reading in the ECC mode, the ECC bits are automatically treated as check bits and are not accessible to the program. It is possible, however, to make the error correction bits available to the program by reading in the no-ECC mode information that has been recorded in the ECC mode. In this case, the exchange treats the ECC bits as data. By doing this, the original data arrangement in words, however, is lost, since one ECC byte is interspersed before every eight bytes of the original data, and eight words recorded in the ECC mode will occupy nine core storage words when read in the no-ECC mode.

External Units

Several input-output and external storage devices are described in this chapter. All of these units follow the same general pattern of control.

Standard Features of Input-Output Units

This section describes the features that are common to most input-output units. These features include keys, lights, and control codes used with the CONTROL instruction. In addition to the common features listed in this section, some units have special keys, lights and control codes applicable only to that type of unit. These special features are described in the sections pertaining to these units.

The labeling of some of the keys and lights described in this section may differ somewhat among the units in order to reflect more accurately the exact function they perform on those units.

Keys, Switches, and Lights

Power On, Power Off. Depression of these keys turns the power in the unit on or off.

Master Power. This is a toggle switch for emergency use. When the switch is turned to the OFF position, all incoming power in the unit is turned off by releasing the main circuit breakers for the unit. To turn power on, the switch has to be turned to the ON position and the circuit breaker has to be manually reset. Normally, the power for the unit is controlled by means of the power-on and power-off keys.

Start. Depression of this key readies the unit for operation. If the unit is not loaded, the material (cards, tape, and so on) is fed into position for reading or writing. When all conditions are met, the unit assumes ready status and a channel-signal indication is sent to the computer. In addition, depression of the start key always extinguishes the check light if the light is on at the time the key is depressed.

If the conditions for operation are not met, the unit remains in the not-ready status and no channel-signal indication is given. The material, however, is still fed into reading or writing position. Depression of the start key causes a channel-signal indication only when, as a result of the depression, the unit changes from not-ready to ready status.

Depression of the start key at a time when the unit is already in the ready status has no effect on the status and operation of the unit except that the check light is turned off. No channel-signal indication is given.

Stop. When this key is depressed, the unit is stopped and is placed in the not-ready status. No more instructions are accepted from the computer until the start key is pressed. If the unit is in operation at the time the stop key is pressed, the unit finishes the current operation before stopping and becoming not ready, unless the operation is a multiple-block data transfer. In the latter case, the operation is interrupted and the unit becomes not ready upon the completion of the current cycle. On units that cannot interrupt their operation without losing some information, depression of the stop key will cause a unit-check indication to be sent to the computer.

Signal. Depression of the signal key when the unit is ready generates a channel-signal indication which is sent to the computer to interrupt the program as soon as conditions permit. If the key is pressed during the execution of an instruction by the unit, the channel signal is automatically delayed by the exchange and is sent to the computer together with the indication that the operation has ended.

Unload. Depression of the unload key causes the material (cards, tape, etc.) remaining in the unit to be fed through the unit without reading or writing, provided the unit is not ready. The unload key is inoperative if the unit is in the ready condition. The stop key must then be pressed before unloading.

Power On. This light indicates that the power in the unit is on.

Ready. The ready light is on whenever the unit is in the ready condition. It indicates that the material is loaded, all operating conditions have been met, and the start key has been pressed. The ready light is turned off by depressing the stop key, by running out of material, or by failure to meet all operating conditions. It may be off also during certain operations which are completed after disconnecting from the exchange, such as tape rewinding.

Select. When more than one unit can be connected to an adapter, the select light indicates the unit which has been selected by the preceding LOCATE instruction. The light is on whenever power is on, regardless of whether the unit is ready or not ready.

Check. The check light is turned on under program control by means of a CONTROL instruction. It is turned off by the start key. This light may be used by the program to indicate that a data error has been detected either by various built-in checking circuits (equipment failure) or by the program itself (incorrect source data). The turning on of the check light is not associated with any change in the state of the ready light.

Some units have, in addition to the above program-controlled light, other check lights for automatic signalling of mechanical malfunctioning, such as a feed check light on the card reader. On these units, the program-controlled light is labeled more descriptively as "read check" or "write check" in order to distinguish between the two types of check lights.

Reserved. This light is turned on and off under program control by means of CONTROL instructions. The reserved light may be programmed to indicate to the operator that the unit is in use, whether momentarily operating or not.

Out-of-Material. The out-of-material light is automatically turned on whenever the unit runs out of material and is thus unable to continue operation. The conditions include empty card hopper, full card stacker, no more paper, and so on. The unit is in the not-ready status when the out-of-material light is on. Depression of the start key turns the light off, regardless of whether the condition causing the light to come on is removed.

Control Functions

The following control functions are common to most external units. The functions are initiated by specifying the corresponding code in a CONTROL instruction.

OCTAL CODE	FUNCTION
016	*Reserved Light Off.* The reserved light is turned off.
017	*Reserved Light On.* The reserved light is turned on.
057	*ECC Mode.* The unit is placed in the ECC mode. Subsequent data will be read or written with ECC bits attached.
116	*Check Light On.* The check light is turned on. The light can be turned off only by pressing the start key on the unit.
157	*No-ECC Mode.* The unit is placed in the no-ECC mode. Subsequent data will be read or written without ECC bits attached.

When the initial-program-load key is depressed, or the power in the unit is turned on, the unit is reset to the no-ECC mode of operation, and the reserved and check lights are off.

IBM 729 IV Magnetic Tape Unit

The tape unit described in this section is the IBM 729 IV Magnetic Tape Unit. It runs at a speed of 112.5 inches per second and handles half-inch tape with seven tracks: six data tracks and one parity check track.

The tape may be recorded at one of two densities. The higher density is 556 bits per inch. The lower density is 200 bits per inch and permits the units to handle tape that is fully compatible with that used on the IBM 727 units. By using one of the two densities, it is thus always possible to communicate with any system or auxiliary equipment designed to work with the IBM 727 tape units or any units of the IBM 729 series. One of the two densities is selected under program control. Except for this difference, both kinds of tape units are controlled and programmed the same way.

The information rate of the tape depends upon the recording density. Since the tape can be read and written at two different densities, the following two information rates are available:

RECORDING DENSITY	INFORMATION RATE IN 6-BIT BYTES PER SEC.	AVG. WORD PERIOD IN μS PER 64-BIT WORD
556 bits/in.	62,550	170
200 bits/in.	22,500	474

The maximum word rate of the tape, corresponding to 62,550 six-bit bytes per second, is thus 5,864 64-bit words per second.

All tape units are connected to the exchange through a tape control. Each exchange channel can accommodate one tape control and as many as eight tape units may be attached to the tape control. Only one of the units attached to a channel, however, can be operated at a time. Simultaneous operation requires more than one tape control, each attached to a different channel on the exchange. Tape units which are connected to different controls can operate independently of one another.

The tape control contains a "byte converter" that converts four successive 6-bit bytes on reading into three successive 8-bit bytes for entry into the exchange, or three 8-bit bytes from the exchange into four 6-bit bytes on writing. Incomplete 6-bit bytes at the end of a word are filled with the first bits of the next word so that no gaps are left on tape. Conversely, successive 6-bit bytes on tape appear in consecutive 6-bit groups in storage without gaps, and may straddle storage word boundaries. As a result, the information rate at the exchange in terms of 8-bit bytes per second is 75% of the tape rate in terms of 6-bit bytes per second; the bit or word rates are the same.

If, during writing, the number of bits in the words to be written as one block is not divisible by six, the remaining four or two information bits are combined with two or four zeros, respectively, to form the last 6-bit byte to be written. On subsequent reading these zeros are treated as information bits. Similarly, during reading, if the number of information bits in the tape block is not divisible by 8, the remaining 6, 4, or 2 bits are combined with zeros to form the last 8-bit byte to be sent to the exchange. Thus, it is possible to read an extra word of all zeros.

The seven tracks on tape are usually designated CBA 8421, where C is the parity track and BA8421 are information tracks. When these bits are stored, bit B is in the lowest-numbered, or leftmost, bit position. Successive bytes are stored in the order in which they are read, with the B bit adjoining the 1 bit of the preceding byte: BA8421 BA8421 BA. . . .

Normally, tapes are written and read in the odd-parity mode of checking where the C-bit provides an odd number of 1-bits in the seven tracks. However, since other systems use the same physical tapes with an even parity method of checking, this mode of writing and reading even-parity tapes is provided. Even parity cannot be used for binary data or with an ECC code.

The weight of the tapes indicating the load on the exchange is a function of the information rate and depends upon the recording density. The following weights are possible:

INFORMATION RATE IN 6-BIT BYTES/SEC	BYTE WEIGHT	WORD WEIGHT
62,550	32	17
22,500	9	6

Keys, Lights and Switches

The keys and lights on the tape unit perform functions similar to the standard set previously described. The functions are combined in a different way to agree with keys and lights on the IBM 729 IV tape units.

Power On, Power Off, Master Power. These keys are located on the tape control and control the power for all tape units connected to one exchange channel.

Load-Rewind. If the unit is in not-ready status, depression of this key causes loading of the tape into the vacuum columns and searching for the load point in a reverse direction. If the tape is more than 450 feet from its load point, a high-speed rewind is initiated first, with the tape out of the columns, before low-speed searching for the load point. The key is inoperative if the unit is ready.

Start. If the reel door is closed and the tape has been loaded into the vacuum columns, depression of this key places the unit in ready status. A channel-signal indication is then given. If the start key is pressed when the unit is in ready status already, nothing happens.

Change Density. Depression of this key changes the density mode in the unit. If the unit was set to read and write in the low density mode, depression of this key changes the unit to the high density mode, and vice versa.

Unload. If the unit is not ready, depression of this key removes the tape from the columns and raises the upper head assembly, regardless of the distribution of tape on the two reels. In addition, the UNLOAD key turns off the tape-indicator-on light if the light is on at the time the key is depressed. If the tape is to be unloaded, the tape should first be brought to the load point by pressing LOAD-REWIND before pressing UNLOAD. The unload key is inoperative when the unit is ready; in order to make use of this key, the unit must first be made not ready by depressing the reset key.

Reset. Depression of this key immediately stops any tape operation in progress and places the unit in not-ready status. If a high-speed rewind is in progress, pressing RESET once causes the unit to change to a low-speed rewind; a second depression of RESET will then stop the unit. Note that, unlike the stop key on other units, RESET will stop any reading or writing operation immediately without waiting for the end of the block. When, as a result of the depression of the reset key, an operation other than rewinding is interrupted, a unit check is sent to the exchange.

Address Selection Switch. This is a rotary switch for assigning a selection address to the unit. It is mounted horizontally and is operated by a large knurled disk. Only a small section of the disk protrudes from the panel. The switch has ten positions marked with the numbers 0 through 9, out of which the positions 0 through 7 represent assignable addresses. Positions 8 and 9 are invalid and a unit that has one of these addresses cannot be selected. The address assigned to a unit is displayed on an illuminated translucent band which rotates with the knurled disk.

Power On, Ready, Select. These lights are standard. (The power-on light is located on the tape control.)

High-Low Density. This light indicates the density mode of the tape unit. Either the HIGH or the LOW portion of the light is illuminated.

File Protect On. This light is on whenever the unit is unloaded or is loaded with a reel that does not have the file-protect ring attached. The light is on regardless of the presence of the file-protect ring during rewinding.

Tape Indicator On. This light is turned on whenever the end-of-tape reflective strip is sensed during writing or when the tape breaks. It is turned off by the initiation of rewinding and by the REMOVE END OF TAPE CONDITION control instruction. When this light is on, every WRITE instruction is terminated by end-exception after writing one block. The light can be turned off manually by pressing the unload key. If a LOCATE selects a tape unit whose tape indicator is on and the unit is in write status, the LOCATE is terminated by end-exception (EE) and end-operation (EOP).

Fuse. This light is on whenever a circuit breaker opens a circuit in the tape unit. When this light is on, the unit is not ready and customer engineer intervention is required.

Control Functions

An end-of-operation is given upon the completion of every instruction unless the instruction is of the SEOP type. The turning on of status indicators is described only when the functions do not follow the general rule or are likely to cause some exceptional status indications.

OCTAL CODE	FUNCTION
016	*Remove End of Tape Condition.* The tape-indicator-on light is turned off.
036	*High Density Mode.* The tape unit is set to read or write at a density of 556 bits per inch. When the power in the tape system is turned on, all units are reset to the high density mode.
037	*Low Density Mode.* The tape unit is set to read or write at a density of 200 bits per inch.
056	*Erase Long Gap.* The tape control is set so that the next WRS or WTM instruction (before recording information), places on tape a blank gap of eight and one-half inches.
057	*ECC Mode and Odd Parity.* The tape control is set to read or write data together with ECC bits and with odd parity.
076	*Space Block.* The tape is moved forward one block to the next gap, without reading or writing. An end-of-operation indication is given upon completion of the operation. If the block consists of a tape mark, end-exception is given as well as end-of-operation.
077	*Space File.* The tape is moved forward without reading or writing until a tape mark has been passed. The tape then stops in the gap immediately following the tape mark and an end-exception as well as an end-of-operation indication is given.
117	*Write Tape Mark.* A standard tape mark (0001111) denoting end of file is written on tape.
136	*Rewind.* The tape is rewound. If the tape is more than 450 feet from the load point, a high-speed rewind is initiated first, with the tape out of vacuum columns, before low-speed searching for the load point. At the completion of the operation, the load point is located at the read-write head with the tape loaded in the vacuum columns. The REWIND instruction differs from other control instructions in that the end-of-operation indication is given as soon as the operation is initiated at the unit. Subsequently, while rewinding takes place, the unit is not ready. When the operation is completed, the unit automatically becomes ready and sends a channel-signal indication.
137	*Rewind and Unload.* This function is the same as REWIND except that the unit does not automatically become ready upon completion of the operation. When the load point is reached, the unit raises the upper head assembly and removes the tape from the vacuum columns. No channel signal is given upon completion of the operation.
156	*No-ECC Mode and Even Parity.* The tape control is set to read or write data without ECC bits and with even parity.
157	*No-ECC Mode and Odd Parity.* The tape control is set to read or write data without ECC bits and with odd parity. When the power in the tape system is turned on, the tape control is reset to the no-ECC mode and odd parity.
176	*Backspace Block.* This is the same as SPACE BLOCK except that the tape is moved one block backward.
177	*Backspace File.* This is the same as SPACE FILE except that the tape is moved backward until a tape mark has been passed. The head then is positioned on the gap immediately preceding the tape mark, so that the next READ operation will read this tape mark.

Operation of Tape under Computer Control

All information on tape is arranged in blocks, one block normally being separated from the adjacent one by a blank gap of tape approximately ¾ inch long. Consequently, reading in the single-block mode transfers the information located between two consecutive gaps. When the end of the last block of an operation is reached, the tape control causes an end-of-operation indication.

A gap, however, can be longer than ¾ inch. Such a gap can be obtained by means of the ERASE LONG GAP control instruction. When a longer gap is encountered during reading, the tape is automatically moved ahead until data are reached. When a READ instruction or an instruction involving forward spacing is given at a point beyond the recorded data, the tape is moved until it is pulled off the reel.

Normally, new data are recorded only so that the new block follows the previously recorded useful data, not by replacing a block located between other blocks. Because magnetic tape involves mechanical devices, the variations in speed and distances are relatively large, and the tape cannot be guaranteed to stop always in the same place in the gap. If these variations happen to be of a cumulative nature, it is possible to damage the following block or leave information that should have been erased. Replacement of a block of data that precedes another useful block is not recommended. This is the reason for the special treatment of all backspacing operations when the preceding operation involving tape motion has been a WRITE. In order to insure that no noise is left on tape in the inter-block gap when the next block is recorded, a BACKSPACE-BLOCK or BACKSPACE-FILE control function, following a WRITE operation, always causes the tape first to be erased forward for approximately half an inch before backward motion is initiated. This erasing does not affect programming in any way, since the erased information follows the last recorded block and therefore is of no consequence to the program.

In the case of rewinding, such erasing does not take place. Consequently, if the tape is rewound following the recording of the last block, it is not safe to add later any more data on the tape without first erasing and rewriting the last block of the preceding run. Normally this last block will be a tape mark, which may be removed when more data are added.

A LOCATE instruction is used to select one of the set of tape units accommodated by a control. Since the maximum number of units attachable to the control is eight, only the three low-order bits of the right effective address, bits 47 through 49, are decoded by the control; the rest of the address is ignored. A tape system connected to a channel may, however, have less than a full set of eight units, or none of the available units may have its address selection switch set to the specified address. If the LOCATE instruction specifies a non-existent address, the previously selected unit is disconnected, but no other unit is selected in its place. As a result, the channel becomes not ready. In any case, regardless of how the LOCATE instruction is executed, an end-of-operation indication is given upon the termination of the operation.

If more than one unit is set to a selection address specified by a LOCATE instruction, all units having this address are selected. The channel is in ready status as long as at least one of the selected units is in ready status. If a READ instruction is issued to a channel having more than one tape unit selected and ready, the information received is meaningless and is accompanied by a unit check that designates data errors. If a WRITE instruction is issued under the same conditions, data are recorded on all units but the check reading phase causes a UK and EOP indication regardless of whether or not actual errors have occurred. The system is not designed for simultaneous operation of more than one unit, and hence such operation is not recommended. If more than one unit is selected at a time, but only one of them is in the ready status, the operation is performed only on the unit which is ready. The effect is the same as if it were the only unit selected.

The mode specifying even or odd parity and ECC or no-ECC type of operation is set up in the tape control and applies to all the tape units accommodated by that control. Thus, to switch between a tape unit operating in one mode and another tape unit operating in a different mode, a CONTROL instruction to change modes must accompany each LOCATE instruction to switch units. The mode specifying high or low density, on the other hand, is set up in the tape unit itself. Consequently, this setting in a given unit is not affected by any changes in the recording density that are made on other units on the same control.

The tape always must be read in the same density in which it has been recorded. The same applies to all spacing operations and to the sensing of tape marks. If read in the other density, the data are meaningless and a data error indication results in most cases. Consequently, a tape mark cannot be recognized unless it is read or spaced over in the correct density. Furthermore, whenever a reading or a spacing operation is performed in the high density mode over data that have been recorded in the low density mode, the tape stops after moving over the first byte the head encounters, and the heads may remain over the recorded data. No error indications, however, are given when spacing is attempted in the wrong density mode.

When the power in the tape control is turned on or the initial-program-load key is depressed, the tape control is reset to the odd parity and no-ECC mode of operation and all tape units are reset to the high density mode. In addition, the tape control is reset to select the tape unit having selection address 0.

Error Correction

The tape can be operated either in the ECC (error checking and correction) mode or the no-ECC mode. The selection of one of the two modes is under program control and is accomplished by means of control instructions. When information is handled in the no-ECC mode, 64 bits constitute a word. On writing, only the 64 data bits are recorded, and they form 10⅔ six-bit bytes on tape. On reading, the proper ECC bits are generated by the exchange before the word is placed in storage. Errors are only parity-checked, not corrected.

The ECC mode provides a means of applying the error checking and correction facilities of the exchange to errors introduced in data on input-output units. In the ECC mode, a word recorded on tape consists of 72 bits; 64 of these are the data bits specified by the control word, and the remaining eight are ECC bits which are provided by the exchange. The ECC bits permit automatic correction of single errors and detection of double errors per word and normally are not accessible to the program. A full word on tape thus is recorded in 12 consecutive six-bit bytes, starting with the ECC bits. On reading in the ECC mode, the eight ECC bits are used in the exchange for error checking and correction. All single errors are automatically corrected before sending the word to core storage. If an error is discovered that cannot be corrected, a data error (UK and EOP) indication is given. The parity check bit in the C track is provided in addition to the information received from the exchange and independently of the mode of operation.

Parity Checking

Two types of parity checking are automatically provided by the tape system—vertical check and longitudinal check. The vertical parity check is associated with each 6-bit byte and consists of placing the appropriate check bit on the C track of the tape. The longitudinal checking is done on a block basis and is provided by a special 7-bit byte at the end of each block recorded. Parity checking is provided by the tape system and is independent of the ECC bits which may be provided by the exchange.

The normal use of the vertical parity check is to write a 1 or 0 bit in the C (check) track of the tape so that a number of 1's in all seven tracks is odd (odd parity). On reading, the seven bits in each byte are checked to see that they contain an odd number of 1's. Any configuration of six information bits may be read in the odd-parity mode.

When the tape control has been set to the even parity mode, the C bit is chosen such that the number of 1-bits in all seven tracks is made even. Since a 1-bit must be present in at least one of the seven tracks for a byte to be read, it is not possible in this mode to read all of the 64 possible configurations of the six information bits. Specifically, the byte 000 000, which requires a C bit of 0, would not be detected on reading. Hence, the even-parity mode is unsuitable for binary data, and the 000 000 byte is never recorded.

Two 000 000 bytes are used as an end-of-block mark while writing in the even parity mode. When an end-of-block mark is sent to the tape unit for writing in the even-parity mode, the block is immediately terminated. If this is the last block to be written, the operation is ended. If more blocks are to be written in the multiple-block mode, an inter-block gap is written and a new block is started on tape with the next full word in storage. If an all-zeros byte is the first byte of the tape record, the tape unit will not write a record but generates an exchange program check (EPGK). It does not require that the termination of the block coincide with the end of a word in storage.

In contrast to the vertical check, which can be either even or odd, the longitudinal check is always based on the even count parity. Whenever a block is terminated, an extra seven-bit byte is automatically recorded at the end of the block. Each bit of this byte provides an even count parity of all bits in the corresponding track of the block. Thus, the total number of 1 bits in each of the seven tracks in a block is always even.

Parity checking takes place both during reading and writing. During reading in the no-ECC mode, a data error indication is given whenever at least one vertical or longitudinal parity bit is found to be incorrect. During reading in the ECC mode, the exchange can correct some of the errors and consequently not every error causes an alarm. When a single error occurs in a word, as indicated by the failure of only one vertical check in 12 successive 6-bit bytes, the exchange ECC circuits correct this error and the tape control then suppresses the unit-check indication. However, if more than one vertical check should fail in a given word, the errors are uncorrectable, and unit check and end of operation are given at the end of the block. In order to accomplish this, the tape control keeps a count of the vertical parity failures per word. The longitudinal parity check is not used when reading in the ECC mode.

During writing, parity checking is accomplished by making use of the double-gap read-write head. A tape that is being written passes first over the write gap to record the data and then over the read gap to read what has been written. All recorded information is thus automatically read for checking purposes. If at any time a 6-bit data byte is read back which does not have the correct vertical parity, unit-check and end-of-operation status bits are turned on at the end of the current block. This checking is not applied to the longitudinal check character written at the end of a block. The above checking with possible error indication takes places regardless of whether writing is in the no-ECC or ECC mode.

The setting of the mode of parity checking is irrelevant when spacing or backspacing tape. All data errors in these operations are ignored.

To summarize treatment of tape data errors:

During Writing	
Longitudinal parity error:	Ignored
Vertical parity error:	UK and EOP
During Reading	
No-ECC Mode:	
Longitudinal parity error:	UK and EOP
Vertical parity error:	UK and EOP
ECC Mode:	
Longitudinal parity error:	Ignored
Vertical parity error:	Ignored if only one per word; UK and EOP if more than one per word.

Tape Mark

A tape mark consists of a block containing the single byte 0001111 and the associated longitudinal parity check bits. It is preceded and followed by the regular ¾ inch inter-record gap. The tape mark is written on tape by means of a WRITE TAPE MARK control instruction and is used to indicate the end of a series of blocks (sometimes called a file).

Whenever a tape mark is encountered on a reading, spacing or backspacing operation, an end-exception indication is caused. In case of reading, no information is transferred to core storage and the operation is always immediately terminated. The end-exception in this case is accompanied by end-of-operation unless the EE causes a termination of a multiple read before the required number of records has been read. Since no data are sent to the exchange, the control word in the exchange is not modified. In the case of spacing or backspacing, the end-exception always is accompanied by end-of-operation. If a tape mark (in single record mode) is read, both an EE and an EOP are set into the indicator register. If a tape mark (in multiple record mode) is read, an EE only is set into the indicator register. A more detailed description of the effect of tape marks on these operations is given in the descriptions of the respective control functions.

In order that a tape mark may be recognized in any operation, including spacing and backspacing, the byte must be read and properly decoded. This can be assured only if the reading, spacing or backspacing operation is performed in the same density mode in which the tape mark has been recorded. Otherwise the tape mark may appear as a regular block of data.

The vertical parity bit accompanying the tape mark is always recorded in the even parity mode. On reading, however, all error indications due to incorrect vertical or longitudinal parity check bits in the tape mark are suppressed. Thus the mode of parity checking set up in the control unit does not apply to the recording and checking of tape marks. When a parity error (resulting from a READ WHILE WRITING operation) occurs, the unit check indication is suppressed.

It should be noted that the byte 0001111 may occur in ordinary data being read, where it is not interpreted as a tape mark. The distinction between a tape mark and a data block is that the tape mark is a single-byte block. Data blocks must contain more than one byte.

Defective Tape

Persistent unit-check indications, after repeated attempts to write a given block, usually indicate some imperfections on tape. The ERASE LONG GAP control instruction is provided to permit skipping over the defective area. This control function turns on the erase trigger in the tape control unit. When this trigger has been turned on, the next WRS or WTM instruction causes an eight-inch gap to be erased on tape before recording the information specified in the associated control word. This eight-inch gap is in addition to the regular inter-record gap. A WRITE instruction executed for a tape unit at load point results in an effective ERASE LONG GAP. The erase trigger is always reset by the next instruction that involves tape motion; if this instruction is not WRS or WTM, the trigger is ineffective.

Normally, backspacing a block and then writing the long gap should place the imperfection within a gap where it will be ignored during subsequent reading. If the block containing the imperfection is more than eight inches long, more than one gap may have to be written so as to cover the required distance of tape. This can be accomplished by repeating the sequence BACKSPACE, ERASE LONG GAP, and WRITE.

Beginning and End of Tape

Both the load point (beginning of tape) and the physical end of tape are marked by reflective aluminum strips. These strips are attached to the back side of the tape and are photoelectrically sensed by the tape unit.

The load point reflective strip is used to position the tape during loading and rewinding. Upon completion of either of these operations, the tape is positioned so that the whole usable tape is ahead of the read-write heads. The tape cannot be moved by any means under program control beyond the load point. Any backspace instructions given with the tape in this position are terminated with an exchange-program-check indication.

The end-of-tape reflective spot is used to indicate during writing that the end of the tape is approaching. It is usually placed five to ten feet before the physical end of the tape. When the end-of-tape reflective strip is sensed, the tape-indicator-on light is turned on and the operation is terminated with an end-exception indication upon the completion of the current block. If this is also the last block to be written with the WRITE operation, end-of-operation accompanies end-exception; otherwise end-of-operation is absent. The unit remains ready so that control instructions may be given to write a tape mark and then to rewind or backspace. The end-of-tape reflective strip is ignored during reading and spacing, and the program must rely upon tape marks to indicate the extent of the useful recorded information.

The tape-indicator-on light stays on until it is turned off either by the initiation of rewinding, by giving REMOVE END OF TAPE CONDITION control instruction, or manually. As long as the light is on, all WRITE instructions given to the tape unit are terminated by end-exception after recording one block. The effect is the same as if the end-of-tape reflective strip had been sensed in the first block. However, it is not advisable to write more blocks once the reflective strip has been reached in order to avoid pulling the end of the tape off the reel. When the tape is pulled off the reel, either by writing beyond the end-of-tape mark or by reading or spacing beyond the recorded data, a unit-check indication is given and the unit becomes not ready. Operator intervention is subsequently required.

File Protection

File protection on the IBM 729 IV units is accomplished by means of a special ring on the tape reel. When this ring is absent, a pin on the tape drive rides freely in the groove normally occupied by the ring, and a writing interlock is automatically set so that it is impossible to record any information on tape. WRITE and WRITE TAPE MARK instructions issued to a tape unit with the tape reel in the protected status are not executed and cause an EPGK indication. A tape unit in protected status, however, can execute READ instructions and all control instructions except WRITE TAPE MARK. The absence of the file protection ring on the reel mounted on a unit is indicated by turning on the file-protect-on light on that unit.

Channel Signal

The channel-signal indication on tape units is caused whenever the unit changes from not-ready status to ready status. This change can be the result of either a depression of the start key or the completion of a REWIND control operation. There is no signal key on the tape units to generate a channel-signal indication.

When more than one tape unit is attached to a single control unit, a channel-signal indication may be given by any of the units regardless of whether or not the unit is currently selected by a locate instruction. No distinction is made between channel-signal indications coming from different tape units attached to the same control unit.

Magnetic Tape Timing

The main physical parameters affecting the timing of READ and WRITE operations on tape are tape speed and the spacing between the read and write gaps of the head. As mentioned previously, the speed of the tapes in the 729 IV units is 112.5 inches per second. The spacing between the read and write gaps is responsible for the difference in the delay of the execution of READ and WRITE instructions. This spacing is 0.3 inches. Since the tape system is not buffered, data transfer between the unit and the exchange is concurrent with the reading or writing at the unit, and hence the following delays apply to the operation of the unit as well as to the data transmission.

When a WRITE instruction is initiated, the recording of information commences five milliseconds after the instruction is received at the unit. This delay does not depend upon whether the tape is moving or is at a standstill at the instant the instruction is received. When a WRITE instruction is completed, the end-of-operation indication is initiated by the unit 136 microseconds after the last data byte of the recorded information is read for checking purposes by the read gap. Because of the spacing between the read and write gaps, this is 2.8 milliseconds after the last byte has been recorded. The tape runs at full speed for about 0.4 milliseconds after generating end-of-operation and stops within 2.0 milliseconds after EOP.

On reading, the tape unit is ready to sense the recorded data three milliseconds after accepting the

READ instruction from the exchange. The time at which reading actually commences, however, depends upon when data are available. This in turn is a function of the length of the inter-block gap. When the gap has been produced by a WRITE operation following a WRITE operation, the gap is somewhat longer than the average ¾ of an inch, and the first data pass the read gap approximately 7.7 milliseconds after the instruction is received at the unit. The gap produced by a write-after-read sequence is slightly less than ¾ of an inch and in this case the delay between the acceptance of the instruction at the unit and the encountering of the first data on tape is approximately 6.7 milliseconds. These times apply to the case when the tape is stationary at the time the instruction is received at the unit and the preceding operation was reading or forward spacing. Reading after writing is not recommended since it cannot be guaranteed that the tape after writing will have stopped in the proper place in the gap. It should be noted, however, that the above times are only approximate, since they depend upon mechanical devices and thus can vary from day to day and from unit to unit.

The end-of-operation at the completion of a READ operation is initiated at the unit 136 microseconds following the reading of the last data byte. The tape subsequently runs at full speed for about 1.4 milliseconds. It stops within 3.0 milliseconds after EOP. When the multiple block mode of reading or writing is used, the tape traverses the inter-block gap at full speed.

The timing considerations of READ apply also to the control functions which involve forward spacing or recording of information. These are the SPACE BLOCK, SPACE FILE, and WRITE TAPE MARK control functions. On forward spacing files, the tape does not slow down in the inter-block gap.

On backspacing, the timing depends upon the preceding instruction involving tape motion. When backspacing follows READ or another forward or backward spacing operation, the tape unit is immediately set in the backward status and the read gap encounters the first recorded byte, which in this case is the longitudinal parity check byte, approximately 11 milliseconds after the instruction is received at the unit. When backspacing follows WRITE, the tape is first moved forward for approximately five milliseconds to erase the information ahead of the head before backward motion is initiated. Because of this delay, the longitudinal check byte is encountered only approximately 25 milliseconds after the backspace instruction is received at the unit. The EOP indication indicating the completion of the operation is generated in either case only after the tape has come to a full stop. This is 16 milliseconds after passing over the beginning of the block. The speed of the backward motion is the same as that during forward motion, or 112.5 inches per second.

The above timing of backward motion applies both to backspacing blocks and files, except that on backspacing files the EOP indication is given only after moving over a tape mark. It should be noted, however, that on backspacing files, unlike on forward spacing files, the tape comes to a stop after spacing over each block. Thus, the total time for backspacing over a file is the same as the sum of the times required to backspace over each block separately, provided the time to issue the instruction is negligible.

Any time after EOP has been received for an operation involving forward motion of tape, a new instruction can be issued to the unit. If the next instruction involves moving tape in the same direction in which it is already moving, the motion continues at full speed. If the next instruction is received while the tape is slowing down, it is immediately accelerated again. In case of write operations, the inter-block gap is made the same length regardless of whether or not the tape comes to a stop.

The execution time for all control functions controlling lights or the mode of operation in the tape control (remove end of tape condition, erase long gap, ECC mode and odd parity, no-ECC mode and even parity, no-ECC mode and odd parity) is 28 μs. This applies also to the LOCATE operation selecting a unit on a tape control. The control functions specifying the density of recording (high density mode and low density mode) apply to the tape unit and the execution time for these instructions is approximately 10 milliseconds. These execution times do not depend upon whether or not the instructions are superfluous.

In the case of the REWIND and the REWIND AND UNLOAD control functions, end-of-operation indication is initiated by the unit approximately 20 milliseconds after the acceptance of the instruction, which is the time necessary to initiate rewinding. The unit subsequently is not ready. The time to complete the rewinding is approximately 1.2 minutes for any tape lengths between 450 and 2400 feet. For tape lengths less than 450 feet, the tape is not unloaded and rewinding takes place at the regular tape speed of 112.5 inches per second. When a REWIND or REWIND AND UNLOAD instruction is issued with the tape already at the load point, the EOP indication is initiated at the unit 28 microseconds after the acceptance of the instruction with the channel signal preceding the end-of-operation by eight microseconds.

All of the above timing considerations as well as the execution times given for the control functions do not include the instruction processing time in the exchange. The above execution times are defined as the time intervals between the instants the instructions are received at the units and the times the units signal to the exchange the completion of the operations.

Manual Operating Procedure

The steps required to load, unload, and otherwise operate a tape unit manually are the same as for any IBM 729 tape units. They will be found in any existing reference manual for systems using these tape units.

Summary of Status Indications

Figure 41 outlines the various status indications, due to exceptional conditions, that are given when an instruction addressed to a tape unit is terminated *by* or *after* sensing the corresponding condition.

Instructions are no-SEOP unless otherwise specified.

IBM 7503 Card Reader

The IBM 7503 Card Reader operates at 1000 cards per minute. Cards are loaded in the hopper face down, 9 edge first. They feed past a first reading station where the checking buffer is loaded, past a second reading station for data entry, and from there to the stacker. A clutch permits intermittent starting and stopping as far as the second reading station; from there the cards feed continuously to the stacker.

At the second reading station, the card contents are transferred to a 960-bit transposition buffer. This buffer changes the information from a row-by-row representation to a column-by-column representation before sending it to the exchange and core storage. After the reading of a card is completed, the card contents occupy up to 960 consecutive bits in core storage. This constitutes a card image. The sequence of bits, in ascending order of bit addresses is:

Column 1, Row 12
Column 1, Row 11
Column 1, Row 0
to
Column 1, Row 9
Column 2, Row 12
Column 2, Row 11
etc., to
Column 80, Row 9

The 12 bits of column 2 are immediately adjacent to the 12 bits of column 1, without any gaps. A complete 960-bit card image can thus be placed within 15 words of core storage, with a 1-bit representing a hole and a 0-bit representing no hole.

There is no pluggable control at the card reader; the central stored program thus has full control on editing of data for greater simplicity and flexibility. The card reader has a byte weight of zero and a word weight of 5.

	CONDITIONS				
Instruction	Load Point	Tape Mark	End of Tape Reflective Strip*	End of Tape Mechanical**	File Protect
Read -- Single Record Mode	---------	EOP & EE	----------	UK	-------
Multiple Record Mode	---------	EE	----------	UK	-------
Write, Write Tape Mark	---------	---------	EOP & EE	UK	UK & EOP
Incomplete Multiple Write	---------	---------	EE	UK	UK & EOP
Space Block, Space File	---------	EOP & EE	----------	UK	-------
Backspace Block, Backspace File	EPGK	EOP & EE	----------	---------	-------
Rewind***	CS & EOP	---------	----------	---------	-------
Rewind SEOP***	CS	---------	----------	---------	-------

* An end-of-tape reflective strip turns on the tape indicator light. A multiple-block write operation is terminated after completion of the first block in which this light is on.
** Tape is pulled off the reel and the unit becomes not ready.
*** The EOP indication for rewind is given immediately upon initiation of the operation. The CS is generated when load point is reached. This can be immediately, if the tape is already at the load point, or upon completion of rewinding.

Figure 41. Status Indication Summary

Keys, Switches and Lights

Power On, Power Off, Master Power, Start, Stop, Signal, Unload. These keys have standard functions. If the stop key is depressed during the execution of a multiple-block read instruction, the operation is interrupted at the end of the current card cycle but no information is lost and, hence, unit check is not sent to the exchange. When the start key is depressed, the original operation is resumed, provided the power has been on continuously.

Motor. This toggle switch controls the power for the feed drive motor and is provided as a safety measure. When the switch is off, the motor is disconnected and the unit cannot be placed in the ready status. Normally, it is on. This switch does not have to be used during the normal operation of the unit.

Power On, Ready, Read Check, Reserved, Out of Material. These lights are standard. The out-of-material light goes on whenever the hopper and feed are empty, or whenever the stacker is full.

Feed Check. This light is turned on automatically whenever misfeeding of cards has made the reader inoperative. When this occurs, a unit-check indication terminates the instruction in progress, if any, and the reader becomes not ready. After clearing the card jam, the feed-check light is turned off by lifting the cards in the hopper and pressing the unload key, thereby running out the cards remaining in the feed.

Control Functions

The following CONTROL instructions, giving standard functions, can be executed by the card reader:

OCTAL CODE	FUNCTION
016	*Reserved Light Off*
017	*Reserved Light On*
057	*ECC Mode*
116	*Check Light On*
157	*No-ECC Mode*

Operation of Card Reader under Program Control

When the card reader is initially loaded, the first card is read and the card image is stored in the transposition buffer. A subsequent READ instruction from the computer then causes the buffer contents to be transferred to core storage. Because the card has previously been read for checking, the check is complete at the time the entire card image has been stored in core storage. In the single-block mode, an end-of-operation is given when the transfer from buffer to core storage is completed, and the absence of unit check at this time indicates that the card now in core storage has been read correctly. A card constitutes a block of information as far as the exchange is concerned.

As soon as the buffer has been emptied, a new card cycle is started to place the next card in the buffer. The timing of the next READ instruction, however, does not have to be related to the state of the buffer. If it is given early, the data transfer is delayed until the buffer is full again. If the next instruction is late, the card reader comes to a stop and waits for the instruction to cause the buffer to empty.

When infrequent READ instructions are given in the single block mode, the unit may have finished the physical moving of the card by the time the next instruction is accepted. If a card jam occurs in such a case, the unit is not ready when the next instruction is issued, and the instruction is rejected. In the multiple-block mode, however, the exchange restarts the unit as soon as the data transfer for the preceding block has been completed. If a card jam occurs after the unit has accepted the instruction, the instruction is terminated with a unit check. Any future instructions then are rejected because of the not-ready status.

Checking of reading is done by comparing the card contents as sensed by the first and second reading stations. This is performed at the time the card image is transferred from the transposition buffer to main core storage. When any uncorrectable errors are discovered, a unit check and end-of-operation are given, and the operation is terminated at the end of the current block. In order to avoid error indications for correctable errors, in the ECC mode a unit check is given and the operation is terminated only when more than one bit per word is read incorrectly.

Error Correction

The card reader can be operated either in the ECC mode or the no-ECC mode by means of the corresponding control instructions.

In the no-ECC mode, which must be used for communicating with devices external to the system, a word consists of 64 bits and occupies 5⅓ columns of a card. When these cards are read, all of the 64 bits are sent to core storage as data, and all of them are accessible to the program.

In the ECC mode, 72 consecutive bits, or six adjacent columns, are sent to core storage as one word. This word consists of the 64 data bits preceded by eight ECC bits. The ECC bits permit automatic correction of single errors and detection of double errors in a word. They are provided during writing by the exchange, and may be punched automatically on the card punch, to provide checking and error correction by means of the ECC checking circuits when cards are read. A data error indication is given if the check is not satisfied and if the error is such that it cannot be automatically corrected. The ECC mode on cards is used primarily to punch binary data for re-use in the system, such as for entry of programs in condensed machine language.

In the no-ECC mode, an entire card appears in core storage as 15 full words. In the ECC mode, a card is treated as 13 full words, corresponding to 78 columns, with the last two columns ignored. When cards are read in the ECC mode with the multiple flag on, and the word count in the control word is greater than 13, a new card is fed as soon as the contents of the first 78 columns of the preceding card have been transferred to the exchange. The last two columns of a card are entered into the buffer and checked, but they are not sent to the exchange in the ECC mode.

Out of Material

The card reader continues to feed cards in response to READ instructions until the hopper and feed are empty or until the stacker is full. Either condition represents an out-of-material status, which is signalled by turning on the out-of-material light and giving an end-exception indication.

When the last card has left the hopper, the reader can take two more normal reading cycles to permit the last card to pass the second reading station and go to the stacker. The third read operation then transfers the contents of the last card from the buffer to storage, whereupon the unit gives an end-exception indication and becomes not ready. If this operation is in the single-block mode, the end-exception indication is always accompanied by the usual EOP. In the multiple-block mode, the EOP indication is given only if this card is the one that would have normally caused end of operation. Otherwise, only end exception occurs, and the program must use a COPY CONTROL WORD instruction to determine how many cards have actually been read with the last READ instruction. When the contents of the last card have been transferred to storage, the card reader becomes not ready. The out-of-material light is turned on at the time the end-exception indication is sent to the exchange.

After the last card has left the hopper, it is advisable not to reload the hopper until that card has gone through to the stacker. (The time this takes depends on the rate at which the program issues READ instructions.) If the last card has gone past the first reading station and stopped before the second reading station, putting more cards into the hopper will turn on the feed-check light and cause a unit check because there is a gap in the sequence of cards which usually indicates a misfeed. The feed will have to be cleared and the last card reloaded to continue operation.

When the stacker becomes full, the out-of-material status is assumed after the execution of the next READ instruction, i.e., the instruction following the cycle in which the stacker-full condition is sensed. Thus, the out-of-material light and the end-exception indication are turned on after transferring to core storage the contents of the card responsible for tripping the sense switch in the stacker. If the stacker becomes full while the last card from the hopper is passing through the feed, the out-of-material condition is delayed until that last card has been read and stacked.

Card Reader Timing

A card cycle of the reader is 60 milliseconds long. The transfer of the card image from the reader's buffer to the exchange takes place at the rate of one 8-bit byte per 62 microseconds and requires a total of 7.5 milliseconds. Defining the card cycle as the interval between the completion of the mechanical reading of two successive cards, during continuous operation the data transfer has to be completed within the first 9 milliseconds of the cycle. Normally, it occurs within the first 7.5 milliseconds. The last 33.5 milliseconds of the cycle are used for reading the following card, i.e., for loading the contents of the card into the buffer. Timing for one of a set of consecutive single block operations is shown in Figure 42. (The cross-hatched areas represent the actual data transmission time.)

The end-of-operation indication signifying the completion of an operation is given immediately upon completion of data transfer, which is near the end of the first 9 milliseconds of the cycle. If the next READ instruction is given any time before the end of the current card cycle, the data transfer for that instruction is executed during the first 7.5 milliseconds of the following card cycle and the card reader operates at the rate of 1000 cards per minute. In order to maintain operation at the rated speed, it is necessary, however, that the data transfer from the buffer to the exchange be completed within the first 9 milliseconds of a cycle. If the buffer is not cleared within this time, either because

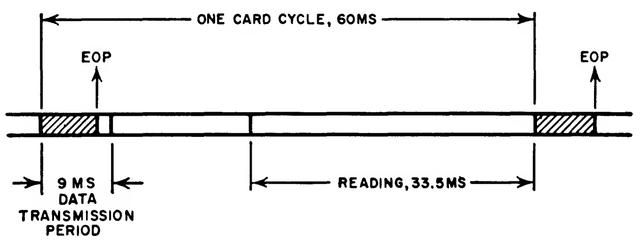


Figure 42. Card Reader Timing

of a delay in issuing the READ instruction or because of excessive load on the exchange, the feed clutch does not engage, and the next card is not fed through the reading station within the current cycle. Consequently, the reader misses one whole card cycle.

When reading of a number of cards is first initiated, the first card image is immediately transferred to the exchange, since it already appears in the card reader buffer. The transfer of the contents of the second card, however, can have an indeterminate delay of up to 60 milliseconds in addition to the usual delay within a card cycle, depending upon the time in the card feed cycle at which the instruction is given. From the third card on, the operation is synchronized and follows the timing as described above. If the feed motor is stationary at the time a READ instruction is issued, an additional delay of approximately 230 milliseconds is encountered for starting and accelerating the motor. The motor is normally turned off by a timer if no READ instructions are issued for approximately one minute.

The time required for the execution of each of the five CONTROL instructions is between 62 and 117 microseconds, depending upon the time of the card reader clock at which the instruction is given. This time does not depend on whether the operation is superfluous or not.

The above execution times do not include the instruction processing time in the exchange, but only the interval between the time instructions are received at the unit and the time at which the unit signals to the exchange the completion of the operation.

Manual Operating Procedure

1. LOADING CARDS

   Place the cards in the hopper.
   Press the start key.

2. RELOADING

   If the hopper is not completely empty, add the new cards to those in the hopper.

   If the hopper has become empty, wait until the last card is stacked and the reader stops with the out-of-material light on. Then proceed as in 1. After the out-of-material status has been reached, reading of any additional cards requires that a new READ instruction be issued.

3. EMPTYING STACKER

   If the reader stops with the out-of-material light on, check to see if the stacker is full. If so, remove cards and press the start key. In order to continue the operation, a new READ instruction must be issued.

4. STOPPING TEMPORARILY AND RESTARTING

   To stop the reader while it is operating, press the stop key. The ready light will go out and the reader will stop. No indication concerning this interruption will be sent to the computer.

   To restart, press the start key. The ready light will come on and the reader will resume the interrupted operation without any intervention by the program. The channel signal indication caused by the depression of the start key will be presented to the computer upon the termination of the READ operation.

5. FEED CHECK LIGHT ON AND READER STOPPED.

   Remove all cards from the hopper.
   Remove the cause of the misfeeding.
   Press the unload key to turn off the feed-check light and to remove any card left in the feed.
   Place it ahead of those removed from the hopper.
   Replace the cards in the hopper.
   Press the start key.

6. READ CHECK LIGHT ON

   Since the read-check light is turned on by the program, its interpretation depends upon the meaning assigned to it by the program. The light may be turned on to indicate that the built-in checking circuits have detected uncorrectable errors in the last card read during the execution of the preceding read instruction, or the program has found the data to be incorrect.

   Whenever the unit is stopped because of data errors discovered by the card reader, the card causing the interruption is the last card fed to the stacker.

7. RESERVED LIGHT

   This light is defined entirely by the program. When the reserved light is on, cards should be loaded only as required by the program currently using the reader. When the reserved light is off, the program has finished the job to which the reader was previously assigned, and the reader may be assigned to a new task.

   Reading of a file of cards is accomplished as follows. Cards are loaded and depression of the start key causes ready and channel signal conditions to be set. If a feed check prevents ready, the channel signal is not set. Channel signal should be recognized by the program as a signal to copy control word. If the reader is ready, the reserved light is turned on and reading of the file begins.

   Channel not ready reject, after a normal EOP, indicates that the operator has stopped the reader before the hopper is empty. End exception with EOP indicates the hopper is empty or the stacker is full (out-of-material condition) and operation is ended. The start after operator stop, or out-of-material stop, turns on channel signal. Ready can be treated as the initial start for single card reading. For multiple card reading in the operator stop condition, the operation was not ended and channel signal does not come on until operation is ended otherwise.

When the last card of the file has been read as indicated by out-of-material light and not ready, the operator should depress the signal key to indicate the logical end of file. This causes the channel signal to be set even though the unit is not ready. By use of copy control word, the program can recognize this condition as end of file. If the program turns off the reserved light at this time, the operator knows that any further channel signals will be interpreted as part of the next file and that it is safe to ready the reader.

IBM 7553 Card Punch

The IBM 7553 Card Punch operates at 250 cards per minute. Blank cards are loaded in the hopper face down, 9 edge first. They feed past a punching station, a reading station for checking, a blank station, and from there to the stacker. Another blank station precedes the punching station, so that when the punch is loaded there are two unpunched cards between the hopper and the punch dies. A clutch permits intermittent starting and stopping as far as the reading station; from there the cards feed continuously to the stacker. An offset device permits cards to be slightly offset in the stacker for easier identification; this is done under program control.

The information to be punched must be assembled in core storage as a card image occupying up to 15 words in 960 consecutive bit positions: column 1 – row 12, column 1 – row 11, column 1 – row 0, to column 1 – row 9, column 2 – row 12, column 2 – row 11, etc., to column 80, row 9.

Keys, Lights, and Switches

Power On, Power Off, Master Power, Start, Stop, Signal, Unload. These controls are standard. If the stop key is depressed during the execution of a multiple-block WRITE instruction, the operation is interrupted at the end of the current card cycle. No information is lost, however, and hence unit-check is not sent to the exchange. When the start key is depressed, the original operation is resumed, provided the power has been continuously on.

Motor. This toggle switch controls the power for the feed drive motor and is provided as a safety measure. When the switch is off, the motor is disconnected and the unit cannot be placed in the ready status. Normally, it is on. This switch does not have to be used during the normal operation of the unit.

Power On, Ready, Write Check, Reserved, Out of Material. These lights are standard. The out-of-material light goes on whenever the hopper is empty or whenever the stacker is full.

Feed Check. This light is turned on automatically whenever misfeeding of cards has made the punch inoperative. When this occurs, a unit-check indication terminates the instruction in progress, and the punch becomes not ready. After the card jam is cleared, the feed-check light is turned off by lifting the cards in the hopper and pressing the unload key, thereby running out the cards remaining in the feed.

Control Functions

The following control functions initiated by CONTROL instructions can be executed by the card punch:

OCTAL CODE	FUNCTION
016	*Reserved Light Off.* Standard function.
017	*Reserved Light On.* Standard function.
056	*Card Run-Out.* The card punch feed goes through two mechanical cycles and feeds two cards, without any punching taking place.
057	*ECC Mode.* Standard function.
116	*Check Light On.* In addition to turning the write-check light on, this control code causes a card to be offset in the stacker.
157	*No-ECC Mode.* Standard Function.

Operation under Program Control

Each single-block WRITE instruction given to a card punch causes one card image to be transferred from core storage to the transposition buffer in the punch. The feed mechanism is then started to punch the buffer contents into the card waiting just ahead of the punching station. Simultaneously with the punching of the new card, the card punched in the preceding cycle is fed past the reading station for checking. Unlike the case of the card reader, the end-of-operation indication is not given until the *end* of the card cycle. At this time the checking of the preceding card has been completed. A card constitutes a block of information as far as the exchange is concerned.

Since the checking of punching takes place only in the subsequent cycle when the card passes the reading station, the card just punched is not yet checked at the time the end-of-operation indication is given. However, in addition to punching errors, errors in a card can be introduced also during transmission of the data from the exchange to the unit. These errors are always discovered during the current cycle. Thus, a unit check accompanying end-of-operation means that either a punching error has been discovered in the card preceding the last one, or a transmission error has occurred in the data punched in the last card. No error indications are ever suppressed because of operating in the ECC mode.

When a multiple-block WRITE is given, more than one card can be punched with a single instruction. A new card image is automatically transferred to the transposition buffer as soon as the preceding cycle has been completed. Whenever an error is discovered, the operation is terminated at the end of the current cycle, and both the unit-check and end-of-operation indications are given. As in the case of the single-block mode, the error could be a transmission error in the last card punched or a punching error in the card preceding the last one.

The reason for delaying end-of-operation beyond the portion of the card cycle when the buffer is loaded is to give the earliest opportunity for catching any errors. The checking of a card is completed only when the whole card has passed the reading brushes, or at the end of the card cycle. By delaying EOP until the end of the cycle, it is possible to notify the computer of any errors in punching upon completion of the next cycle instead of waiting until the third card is punched. Construction of the punch does not permit verification in the same cycle.

When the punch is initially loaded and is set into the ready status, blank cards are fed into the punching station, the blank station preceding the punching station, and also into the checking station following the punching station. Since the card fed into the checking station is never punched, the deck of cards punched immediately after the manual loading is always preceded by a blank card.

When a card jam occurs, the unit becomes not ready, the feed-check light is turned on, and the current operation is terminated with a unit check. Unlike the case of the card reader, a misfeeding of cards can occur only while the unit is executing an operation, and it is always indicated by means of unit check.

Error Correction

The card punch can be operated either in the ECC mode or the no-ECC mode by means of the corresponding control instructions.

In the no-ECC mode only the 64 data bits of a word are punched, and 15 core storage words fill the 80 columns of a card. This mode is used to punch cards to be used outside the system.

In the ECC mode each word consists of 72 bits, 64 data bits preceded by eight ECC bits. One such word occupies six adjacent columns on the card.

In order to provide an integral number of words per card, only the first 78 columns of the card can be punched in the ECC mode, the last two columns being left blank. Consequently, when a sequence of cards is punched in the multiple-block mode with a single control word, each card contains 13 words in the first 78 columns and two blank columns. The last two columns are not used, and it is important that they be blank before punching because all 80 columns are checked at the following reading station. Any prepunching in these columns causes the check to fail.

Out of Material

The card punch continues to feed cards in response to WRITE instructions until the hopper becomes empty or until the stacker becomes full. Either condition represents an out-of-material status. It is indicated by turning on the out-of-material light and giving an end-exception indication.

As the last card leaves the hopper, the current WRITE operation is executed normally except that it is terminated by end-exception. The punch then becomes not ready. Because the cards not yet punched are all blank, no attempt is made to empty the feed before indicating the out-of-material condition. As in the case of the card reader, the end-exception indication may or may not be accompanied by end-of-operation.

When the stacker becomes full, the response of the card punch is the same as above, and, unlike the case of the card reader, the end-exception indication is given at the completion of the current operation. The punch subsequently becomes not ready. The out-of-material light is always turned on at the time end-exception is sent to the exchange.

Card Run-Out

When the punching of a set of cards is finished, the last card will not have been checked at the time the last WRITE operation is completed. To check and stack the last card, which at this time is in the checking station, two more card cycles are needed, since the card must pass through the blank station before it reaches the stacker. These two extra cycles may be conveniently obtained by giving a CARD RUN-OUT control instruction. This control instruction causes two blank cards to be fed through the punch. Absence of unit check after this operation then verifies the correctness of the last card actually punched.

An alternative method of feeding a blank card through the punch is by writing one or a few words consisting of all zeros. When this is done, the card punch has to be in the no-ECC mode, since the ECC bits associated with an all-zero word are not zero.

Card Offset

In addition to turning on the write-check light, the CHECK LIGHT ON control function causes the next card to be offset in the stacker. This facilitates locating the card which is in error.

The card that is offset is the next card to be fed to the stacker as a result of a WRITE or a CARD RUN-OUT operation. This is the card that passed the checking station in the last card cycle and currently appears in the blank station between the checking station and the stacker. Consequently, if the CHECK LIGHT ON control instruction is issued after the cycle in which unit check accompanies end-of-operation, and this unit check has been caused by a punching error, the card in error is offset. If the data error responsible for the unit check has been introduced during transmission of the data from the exchange to the unit, the erroneous card is the one following the offset card.

Card Punch Timing

A card cycle of the punch is 240 milliseconds long. Data transfer from the computer to the card punch buffer takes place at the rate of one 8-bit byte per 64 microseconds, thus occupying 7.7 out of the 240 milliseconds. Defining the card cycle as the interval between the instants at which punching of two successive cards is completed, data transfer has to occur within the first 22 milliseconds in order to insure continuous operation. The last 184 milliseconds of the cycle are used for the actual punching of the contents of the buffer into the card. Also, during the last 184 milliseconds of the cycle, the card preceding the one being punched is checked at the reading station. The relative allotment of time within a card cycle for one of a set of consecutive single block operations is shown in Figure 43. (The cross-hatched area represents the actual data transmission time.)

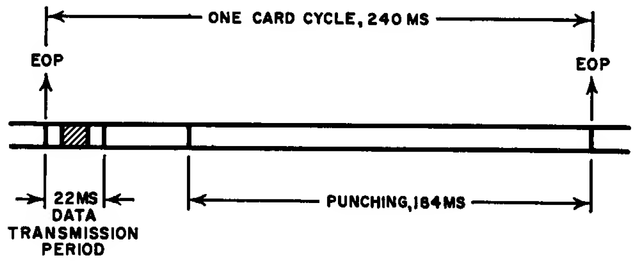


Figure 43. Card Punch Timing

The end-of-operation indication signifying the completion of an operation is given upon completion of the punching, which is at the end of the card cycle. The unit thereupon is in a position to receive a new WRITE instruction. Since the punch buffer at this time is empty, the data transfer associated with the new WRITE instruction takes place immediately after the instruction is issued. If the instruction is issued within the first 14.3 milliseconds of the cycle, data transfer for that instruction can be completed within the first 22 milliseconds of the cycle, and the punch operates at the rated speed of 250 cards per minute. If the instruction is delayed and the buffer is not loaded within the 22 milliseconds, the feed clutch is disengaged. The motor is turned off if no instructions are issued to it for approximately three minutes.

When the punching of a card is initiated with the motor stationary, the completion of the first WRITE instruction may encounter an indeterminate delay of up to approximately 300 milliseconds in addition to the usual delay within a card cycle, in order to accelerate the motor and engage the feed clutch. From the second card on, the operation is synchronized and follows the timing as described above. Note, however, that regardless of the above delay, the data transfer from the exchange to the punch associated with the WRITE instruction follows immediately after the instruction is issued.

The card cycle initiated by a CARD RUN-OUT control instruction has the same format as the one described above, except that no data transfer takes place. Because of this, the card is fed in the current cycle if the instruction is issued at any time within the 22-millisecond period following the preceding end-of-operation. As in the case of WRITE, the EOP indication for the CARD RUN-OUT is given upon the completion of the card cycle.

The time required for the execution of the RESERVED LIGHT OFF, RESERVED LIGHT ON, ECC MODE, NO-ECC MODE and CHECK LIGHT ON control instructions is 80 microseconds. This time does not depend on whether the operations are superfluous or not.

None of the above execution times includes the instruction processing time in the exchange but only the interval between the time instructions are received at the unit and the time the unit signals to the exchange the completion of the operation.

Manual Operating Procedure

The rules for manual operation of the card punch are analogous to those of the card reader described previously.

However, unlike the case of the card reader, it is not always possible to identify the card responsible for a unit check. When the card punch stops because of a data error, the erroneous card is either the one last fed to the stacker or the following one, depending upon the cause of the error.

IBM 1403-2 Printer

The IBM 1403-2 Printer has 132 printing positions and operates at a speed of 600 lines per minute. Each printing position is capable of printing any one of 49 different symbols, which include numerical, alphabetic and special characters and the blank space. The printing density is 10 characters per inch. A chain consisting of 119 characters and a blank space may also be used. Speed with this chain is 275 lines per minute.

Information to be printed is coded in an 8-bit code, one line occupying up to 17 words. A WRITE instruction then transfers this block from core storage to the printer's buffer, which stores one complete line of printing. When all the information to be printed on one line has been transferred, printing is initiated.

Checking of the printer operation is automatic and concurrent with printing. Each character code sent to the printer is associated with an odd parity bit. These bits are used to detect errors introduced in the transmission of the character codes from the exchange to the printer control as well as to check the storing and handling of these codes within the printer control. In addition, an echo caused by the regenerative part of the hammer firing pulse is used to verify that a hammer has been fired at the proper instant and that it has not been fired at any other time.

Operation of the carriage is under program control and is determined by a control field that must always precede the information to be printed on one line. There are two carriage speeds, the selection of which is automatic and depends upon the distance the carriage has to advance. A carriage control tape is used for restore, automatic overflow, and out-of-materials functions only.

Keys and Lights

Power On, Power Off, Start, Stop. These keys are standard. Depression of the power on key at any time causes a machine reset. If the stop key is depressed during the execution of a single-block WRITE instruction, the key has no effect on the current operation. If the stop key is depressed during the execution of a multiple-block WRITE instruction, the operation is terminated upon the completion of the printing of the current line. No information is lost, however, and hence no error indication is sent to the exchange. When the start key is depressed, the original operation is resumed, provided the power has been on continuously.

Carriage Space. Depression of this key causes the carriage to advance by one line space.

Selective Printing Keys. Six keys, which comprise a horizontal strip switch, are labeled 1, 2, 3, 4, SEL RUN, and RESTORE. These keys provide a selective printing of properly tagged data. With the SEL RUN key depressed, any combination of keys 1, 2, 3 and 4 may be used. Depression of the RESTORE key resets the other five keys to the off position.

Mode Selection Keys. A second horizontal strip switch, composed of keys labeled 6-bit, 8-bit, 48 BCD, 48 ECS, 120 ECS, and RESTORE, allows selection for the mode of operation. The 6-bit and 8-bit keys refer to the incoming byte size, while those labeled 48 BCD, 48 ECS, and 120 ECS describe the chain and related code being used. Only one byte size can be used at any one time. These keys are not interlocked and depression of the RESTORE key resets the others to the OFF condition.

Power On, Ready, Check, Reserved, Out-of-Material. These lights are standard. The out-of-material light comes on when the last form has passed the form-sensing switch.

Forms Check. This light indicates that a form jam occurred at the printer.

Automatic Overflow Key. This is a lighted key when depressed and in an overflow condition. It conditions the carriage for automatic overflow.

Carriage Restore Key. Depression of this key causes the carriage to restore to the first line of the next form.

Carriage Stop Key. Depression of this key allows the operator to stop the carriage if it is in a runaway condition.

Control Functions

Standard control functions initiated by a CONTROL instruction and executed by the printer are:

OCTAL CODE	FUNCTION
016	*Reserved Light Off*
017	*Reserved Light On*
116	*Check Light On*

Carriage Control

Operation of the carriage is under the control of a special control code, called CARRIAGE CONTROL FIELD. The carriage control field consists of one or more 8-bit bytes and specifies the number of lines over which the carriage must skip before reaching the line on which the information immediately following is to be printed. Every line to be printed must be preceded by a carriage control field.

After printing, the carriage automatically advances by one line position. This post-spacing feature saves time because it permits the carriage to advance before the next carriage control field has arrived. If the next carriage control field specifies skipping additional lines, the carriage continues to move to the desired line. If the next carriage control field specifies to skip 0 lines, the previous post-spacing operation results in single spacing. If the carriage control field is delayed, the carriage waits at the next line. The post-spacing, however, can be suppressed when the next line is to be printed on the same line position as the present line. A SUPPRESS POST-SPACING carriage control function is provided for this purpose.

The carriage control field permits skipping up to 223 lines in one operation. This skipping command is encoded in one 8-bit byte, and if only normal skipping is desired, the carriage control field does not have to have more than one 8-bit byte. In addition to the skipping codes, however, the carriage control field can specify a number of other functions. In the latter case more than one byte is necessary to specify carriage operation.

Figure 44 shows the coding used in the 8-bit bytes of the carriage control field.

Carriage control bytes with decimal codes 0 through 223 specify skipping up to 223 lines in one operation. A skip byte is always the last byte of the carriage control field, and the printer interprets the following information as data to be printed. The remaining carriage control bytes with decimal codes 224 and up provide for special functions. These codes, except for the SKIP TO RESTORE code, cause the following byte to be interpreted as another carriage control byte, thus making it possible to append line selection information to the control bytes specifying special functions.

Code			
Binary	Decimal	Meaning	
0000 0000	0	Skip 0 lines (single space)	
0000 0001	1	Skip 1 line (double space)	
0000 0010	2	Skip 2 lines (triple space)	
.	.	.	
.	.	.	
1101 1111	223	Skip 223 lines	
1110 0000	224	Select No Report	And suppress post-spacing if printing is suppressed
1110 0001	225	Select Report 1	
1110 0010	226	Select Report 2	
1110 0011	227	Select Reports 1 or 2	
1110 0100	228	Select Report 3	
.	.	.	
.	.	.	
1110 1000	232	Select Report 4	
.	.	.	
.	.	.	
1110 1111	239	Select Reports 1,2,3 or 4	
1111 0000	240	Suppress Post-Spacing	
1111 0001	241	Skip To Restore	
.	.	.	
.	.	.	
1111 1101	253	Not Used	
1111 1110	254	End	
1111 1111	255	No Operation	

Figure 44. Carriage Control Field Codes

Carriage control bytes that initiate functions other than skipping are:

1. Codes 224 through 239 operate in conjunction with the report selection keys on the printer and permit selective printing of data.

2. Code 240 suppresses post-spacing. It allows normal carriage operation before printing, but the automatic spacing after the printing cycle is omitted. The suppression of post-spacing makes it possible to print two blocks of information on one line position. In order for the next line to print in the same line position as the present line, it must have the skip 0 lines carriage control byte.

3. Codes 242 through 253 do not have any functions assigned to them and are ignored by the printer. They have the same effect as the no operation code. For compatibility with other devices, it is recommended that these codes not be used.

4. Code 254 is the end code. It causes the subsequent bytes to be ignored and terminates the current block of the operation. Code 254 in the carriage control field suppresses all printing. Carriage control field bytes preceding END code are decoded and the functions are executed. For a WRITE operation to cause no action at all at the printer, the end code must be preceded by a suppress post-spacing code and followed by a skip 0 lines code. The end code itself does not suppress post-spacing or end the carriage control operation.

5. Code 255, no operation, is ignored by the printer and may be used as a filler preceding another carriage control byte.

The line spacing can be either one-sixth or one-eighth of an inch long, depending upon the manual setting of the feed clutch. The above carriage control fields apply to either case, the line space being always defined by the current setting of the clutch.

An overflow feature is provided on the carriage control of the printer. A punch in channel 12 of the paper tape designates the line following the last line on which printing is to appear. Whenever this punch is sensed, a restore is generated which positions the forms tractor to the restored position on the next form. The restored position is designated by a punch in channel 1. This feature is automatic and no signal is given to the program. The overflow feature may be made inoperative by a switch on the printer.

A punch in channel 12 designates the end of printing on the form and should be present under all conditions. When an out-of-material condition exists, the 12 punch designates the end of printing on the last form. Under these conditions, and when the 12 punch has been sensed, an end-exception is generated. An end of operation accompanies the end exception if completion of a current line also constitutes completion of the write operation. Further attempts to write on the printer result in a not-ready status.

The form may be restored to the position marked by the punch in channel 1 by means of a skip to restore (241) in the channel control byte.

Printing under Computer Control

Each line of printing is considered a block of information. This block consists of up to 132 data characters, coded in an 8-bit code, and a carriage control field, consisting of one or more 8-bit carriage control bytes. The carriage control field and a full 132-character line thus occupy at least 1064 contiguous bits or 16⅝ words in core storage. The carriage control field is merely control information; it bypasses the buffer in the control unit and cannot be printed. The eight bits of the byte are numbered 0-7 from left to right. In 6-bit ECS, 6-bit BCD, and 8-bit BCD, the six rightmost bits (2-7) are part of the print code. The one remaining case is 8-bit ECS. In this case the rightmost seven bits (1-7) are part of the print code. If the printer has a 48 character chain, bit 7 (which selects upper and lower case characters) is ignored. Regardless of the printer mode, the printer parity checks the entire 8-bit byte received from the exchange. It is not necessary that the bits which are not part of the print code be zero.

In order to print a full line of 132 characters, a control word with a word count of at least 17 is required. A larger word count may be required when long carriage control fields are used. The last unused bytes, if any, of this block are ignored and are not sent to the unit. When a shorter line is desired, a smaller word count may be specified. In this case, the operation is terminated when the count reaches zero. The count provides a means of controlling the line length in increments of eight characters. If the desired number of characters does not correspond to an integral number of words, the unused byte position can be filled with blanks. Alternatively, the last data character sent to the printer can be followed with an end code, bits 1111 1110. In this case the contents of the count field are irrelevant as long as they permit the end code to be sent to the printer. The end code does not cause printing; it causes the unit to terminate the current block regardless of whether or not the word count has reached zero.

One 132-character line is the maximum amount of information that can be printed with one instruction in the single-block mode. When the multiple-block mode is specified, any number of lines can be printed with one instruction. In this mode, as long as the count is not exhausted, a new block is automatically started with the next full word in core storage. The unused bytes of the preceding word are ignored. A carriage control field, however, has to precede the print information for each line. Consequently, if more than one line is to be printed with one instruction, every word following the one containing the end code or the 132nd character printed has to start with a carriage control field.

When a WRITE operation has been completed as specified, an end-of-operation indication is given. In the single-block mode this indication is sent immediately after the completion of the printing cycle. In the multiple-block mode it is given upon the completion of the last line of the operation. In either case, the operation is considered to be completed before post-spacing, if any, has taken place. The absence of unit-check at this time indicates that all lines specified by the instruction have been printed correctly. Conversely, a unit-check indication always means that one or more characters of the last line have been printed incorrectly. Such a unit check is always accompanied by end-of-operation, regardless of whether or not the operation is completed.

Interlocking between carriage skipping and printing is fully automatic. Printing always starts when the data for the line of printing have been loaded into the buffer and the carriage has come to a stop. The timing of a WRITE instruction, however, does not have to be related to the state of the carriage. If a WRITE instruction causes the buffer to be filled up before the carriage has come to a stop, printing is automatically delayed. Normally spacing takes more time than data transmission and this delay is reflected in the rated printer speed. But if the carriage is already positioned by the time the buffer is filled, printing is initiated immediately.

Selective Printing

Selective printing keys in conjunction with report selection carriage control bytes provide for selective printing of data sent to the printer. This feature makes it possible for the operator to assemble reports out of selected lines of print information without modifying each time the program that provides the data.

Selection of data to be printed is accomplished by matching the four rightmost bits of report selection codes with a bit map generated by depression of the selection keys. Assuming the bits in each byte to be numbered from 0 through 7, starting with the leftmost bit, a depression of select 1 key corresponds to a 1-bit in bit position 7, depression of select 2 corresponds to a 1-bit in bit position 6, depression of select 3 corresponds to a 1-bit in bit position 5 and depression of select 4 corresponds to a 1-bit in bit position 4. Whenever a carriage control byte has bits 1110 in the high-order positions, the printing of the line is suppressed unless there is a 1-bit in the low-order four bits to match the selective printing key that has been depressed. For example, code 225 (1110 0001) permits printing of the current line only if the select 1 key has been depressed while code 227 (1110 to 0011) allows printing for keys select 1 or select 2. On the other hand, when the key select 1 is depressed, only lines with the report selection codes 225 (1110 0001), 227 (1110 0011), 229 (1110 0101), 231 (1110 0111), 233 (1110 1001), 235 (1110 1011), 237 (1110 1101), and 239 (1110 1111) can be printed.

When all print selection keys are off, the report selection codes are ignored. If there is no report selection code in the carriage control field, the key setting is ignored. In either case no selection takes place, and all data are printed. If there is more than one report selection code in the carriage control field only the last line is significant.

When a selective printing key is on and a report selection code is encountered that does not match the key, the remainder of the block, including the following carriage control bytes, is ignored, and printing does not take place. Transmission of data to the printer continues, however, until either the end code is encountered, the control word count is exhausted or the buffer is filled up. When printing is suppressed, post-spacing of the carriage also is suppressed.

Normally end of operation is given when printing of the line is completed. When printing is suppressed, end of operation is sent at the completion of data transfer. No indication concerning the suppression is ever sent to the computer.

Out of Material

The printer continues to print in response to WRITE instructions until the last form has passed the form-sensing switch. When the form-sensing switch has been tripped, the out-of-material light comes on and an end-exception indication is given upon completion of the printing of the current line. An end-of-operation accompanies the end-exception if the completion of the current line constitutes also completion of the WRITE operation.

The punch in channel 12 of the carriage tape is used in conjunction with the forms sensing switch so that the current form will be completed when an out-of-material condition occurs. Because of this feature, the ability to single-line print under manual control is not provided.

Character Codes

The character set that can be printed on the printer consists of 48 distinct non-blank characters and the character blank, all coded in an 8-bit code. This set is shown in the Appendix. Of the non-blank characters, 44 are the same as those that can be printed on the console printer. These and the blank have common codes on the two devices. The remaining four are special characters that are provided on the printer only. This code is known as the 48 ECS set.

Each character code is sent to the printer as an 8-bit byte. If the bits in each byte are numbered from 0 through 7, starting with the leftmost (most significant) bit, the character code occupies the six bit positions 1 through 6. The bit numbered 0 always must have the value zero. Bit position 7, which represents case shift in the console printer, is ignored by the printer in codes for which bits 1 through 6 represent valid characters.

Thus code 0010 1100, which represents character *a* in the console printer code, prints as *A*, and code 0010 0011, which represents the character = in the console printer, prints as $.

When a 1-bit in bit position 0 is encountered or bit positions 1 through 6 specify an undefined code, the code is ignored and does not cause any printing. It has the same effect as the blank code. Its use for this purpose, however, is not recommended for compatibility reasons. In addition to the 48 ECS set, the 120 ECS and 48 BCD sets can be used. They are also shown in the Appendix.

Printer Timing

The time required to print and space one line is about 100 milliseconds. Data transmission from the computer to the printer's buffer always follows immediately after the WRITE instruction is issued and takes place at the rate of one 8-bit byte per 64 microseconds. A complete line of printing information is thus transmitted in approximately 8.5 milliseconds. The time required to space over up to three lines is approximately 20 milliseconds, out of which the first 8.5 milliseconds are overlapped with the data transmission time. The 20-millisecond period can be occupied either by post-spacing due to the preceding print cycle, by skipping for the present cycle or by a combination of both. When spacing is completed, the contents of the buffer are printed. This takes approximately 80 milliseconds. The above timing of the printer is shown in Figure 45.

The end-of-operation indication signifying the completion of the printing of a line is given at the very end of the operation, whereupon post-spacing for this cycle takes place. Any time thereafter a new WRITE

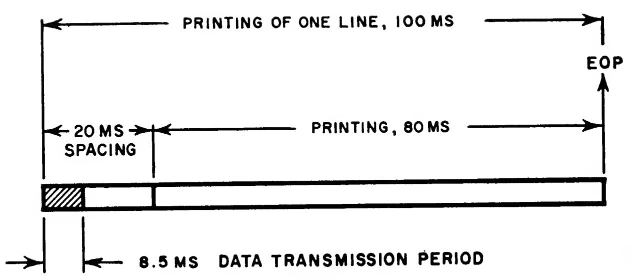


Figure 45. Printer Timing

instruction can be issued, which then immediately initiates repetition of the above sequence of events. The printer does not have any mechanical cycles with which the program has to be synchronized, and hence the printing of a line can start at any arbitrary time. The reduction in speed incurred by delaying the issuing of the next WRITE instruction is due only to the actual delay between the time the EOP of the preceding operation is received and the next WRITE instruction is issued.

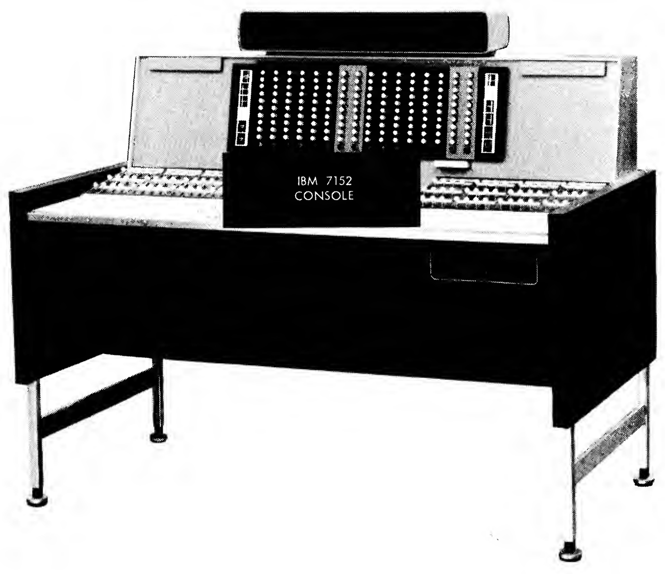


Figure 46. Console Entry and Display Devices

Manual Operating Procedure

The steps required to load, unload, and otherwise operate the printer manually are the same as for the IBM 1403 Printer. Description of this procedure as well as the associated controls can be found in the *IBM 1401 Data Processing System Reference Manual.*

IBM 7152 Operator's Console

The IBM 7152 Operator's Console provides direct communication between operator and computer. For maximum flexibility, the console is treated as an input-output device which is connected to the computer via the exchange. Thus, all functions on the console are fully under stored program control. More than one console may be attached.

The console contains a number of toggle switches, columnar switches, rotary switches, and a keyboard for input. The switches are always scanned to furnish the first three input words when the console receives a READ instruction. Further input words come from the keyboard. A READ instruction can be requested from the program by sending a channel-signal indication.

For output, the console contains lights, a numerical display, and a low-speed printing mechanism. The lights and digital displays are set up by the first three words of the output message sent by a WRITE instruction. Any further words in the output message control the printing. The physical arrangement of the various input and output devices on the console is depicted in Figure 46.

The keyboard and the printing mechanism not only serve for input and output, but together they constitute also a typewriter. All information entered into the computer via the keyboard is also printed. By making the console not ready, the user can use the typewriter for making notes and for adding remarks to the information entered into the computer or typed out by it.

Two keys, signal and enter, cause a channel-signal indication to be sent to the computer. The enter key also prepares the keyboard for entering information. A READ operation following depression of the signal key is terminated by the console when the words of switch inputs have been read. A READ operation following depression of the enter key is not terminated automatically, but will accept characters from the keyboard until the operation is terminated through exhaustion of the control word or by depression of the end or erase keys.

Two computer system operating keys, initial program load and emergency power off, are mounted at the central console for operating convenience but they are not an integral part of every console. Likewise, the central console will contain two lights reflecting the status of the system: a RUNNING and an INACTIVE light. Except for these system controls and the signal and enter keys, none of the keys on the console directly affects the functions of the computer.

Facilities needed for maintenance of the computer are provided separately and are not part of the operator's console. The console has a byte weight of zero and a word weight of 3.

Keys

Power On, Power Off. These keys are standard.

Signal. Depression of this key when the unit is ready generates a channel-signal and disables the keyboard. The key is inoperative when the unit is in the not-ready status.

Ready. If all the conditions necessary for operation are satisfied, depression of this key puts the unit in the ready status. The ready key has the same function as the start key on standard units, except that it does not automatically generate a channel-signal indication.

Not Ready. This key is the same as the stop key on standard units; it places the unit in not-ready status. If the console is reading or writing at the time the not-ready key is pressed, the operation is interrupted, but no interrupt indication is sent to the computer. The operation is resumed again when the unit returns to the ready status.

Enter. Depression of this key *enables* the keyboard and generates a channel-signal indication. If the keyboard has been enabled, it becomes *active* when the subsequent READ operation has completed reading of the three words from the console switches. The active state of the keyboard permits information to be entered thereby and is indicated when the enter light is on and the keyboard is unlocked. A READ instruction received by the console with the keyboard not enabled is automatically terminated when the three words scanned from the switches have been sent to the exchange.

The enter key enables the keyboard and generates a channel-signal regardless of any reading or writing currently taking place at the console. The key is ineffective when the console is in the not-ready status.

End. Depression of this key when the keyboard is active (the enter light is on) enters the END function code, locks and disables the keyboard, and turns off the enter light. In the single-block mode this causes the operation to be terminated with an end-of-operation indication, while in the multiple-block mode subsequent reading is limited to repeated scanning of the console switches. Depression of the key at any other time disables the keyboard if it previously has been enabled, but does not send the end function code or any other signal to the computer. The end key is inoperative when the console is in the not-ready status.

Erase. Depression of this key when the keyboard is active enters an ERASE function code, terminates the READ operation with an end-exception indication, locks and disables the keyboard, and turns off the enter light. During writing on the console printer, the key immediately terminates the WRITE operation with an end-exception indication. In both cases the end-exception is accompanied by end-of-operation unless the multiple block flag in the control word is on. When the keyboard is not active for entry of data or the console printer is not writing, the key is inoperative. The key is inoperative also when the console is in the not-ready status.

Ribbon Shift. Depression of this key causes ribbon shift. If the console printer is equipped with a two-color ribbon, all characters that are entered through the keyboard while the ribbon shift key is depressed are printed in the color associated with the ribbon shift. In addition, when a character is entered through the keyboard with the ribbon shift key depressed, the character code sent to core storage is modified so as to cause ribbon shift when the code is subsequently sent to the console printer for writing.

Lights

Power On, Ready, Check, Reserved. These lights are standard.

Enter. This light indicates that the keyboard is active. The light goes off when the keyboard becomes inactive.

Wait. This light goes on when the signal or enter key is depressed and is turned off by completion of the first three words of a READ operation. It indicates to the user that the computer has not yet accepted the information set up on the switches, which therefore should not be changed. An audible click is generated each time the wait light is turned off.

Control Functions

The console has two sets of control functions. One set is initiated by means of CONTROL instructions, the other during WRITE instructions.

The set of control functions initiated by CONTROL instructions is listed below. These control functions are similar to those used on other input-output units.

OCTAL CODE	FUNCTION
016	*Reserved Light Off*
017	*Reserved Light On*
116	*Check Light On* These functions are standard
177	*Sound Gong.* This function causes a clearly audible gong to sound a single stroke.

The end-of-operation indication signalling the completion of the SOUND GONG control operation is given immediately upon the initiation of the operation, even though the striking of the gong takes place some time later. The unit is subsequently available for another instruction. If a number of SOUND GONG control instructions are given so that the next instruction follows immediately after the end of operation because of the preceding one, the gong sounds at its maximum rate of approximately once per second.

The other set of control functions is associated with the console printer and keyboard. In order to perform a function, the required code has to be transmitted to the console printer by means of a WRITE instruction. When these codes are sent to the console during writing, they perform the indicated function but do not cause any printing. The code can be entered by depressing the appropriate key during reading. The following list describes these codes.

BINARY CODE	INTERPRETATION OF CODE DURING WRITING	KEY THAT GENERATES CODE DURING READING
1111 0111	Blank	Erase
1111 1011	Tabulate	Tabulate
1111 1100	Backspace	Backspace
1111 1101	Carriage Return	Carriage Return
1111 1110	End	End

Binary Keys and Key-Lights

The console has 64 keys, arranged in two banks of 32 each, and grouped by fours within each bank. Associated with each key is a binary key-light that can be set by the output word from the computer or by the key. Each group of keys and key-lights has a frame for a single labeling strip which can be readily changed. The key has a toggle action with a center neutral position and two extreme latching positions. In one extreme position the key turns the light on; in the other extreme position it turns the light off. When latched in either position, it holds the light on or off so that it cannot be changed by the computer. When the key is in the neutral position, the corresponding light can be set on or off by a WRITE instruction.

The bits corresponding to the states of the 64 binary key-lights make up the first input word when a READ instruction is issued and the first output word when a WRITE instruction is issued. The upper row corresponds to bit positions 0 through 31, from left to right, the lower row to bit positions 32 through 63. When a light is on, it corresponds to a one bit; when it is off, to a zero bit.

Numerical Switches

There are two banks of decimal columnar switches. Each bank consists of eight columns, each of which contains keys for the digits 0-9 and a reset key. Each column is encoded with four bits. Digits 0-9 are encoded 0000-1001. When no digit key is depressed, the output for the column is encoded 1111.

Each digit key is illuminated when depressed. The depression of any key releases the other in the same column. There is also a general release bar for each of the two banks.

When a READ instruction is issued, the setting of the 16 numerical switches makes up the 64 bits of the second input word, from left to right, the four bits of the leftmost column occupying bit positions 0 through 3.

The keys in each column are arranged from bottom to top in the order: reset, 0-9. Reset and digits 8 and 9 are distinguished in appearance from digits 0-7 to facilitate entry of octal data. The rightmost two columns in each bank are similarly distinguished from the first six. The banks are grouped together physically.

Numerical Display

A group of sixteen positions is used to display numbers in decimal or octal form. Each position consists of a device for displaying one of twelve characters, or a blank. The twelve characters that can be displayed are the digits 0-9, minus, and decimal point. The character to be displayed at any position is selected by decoding four bits sent from the computer. Codes 0000-1001 are decoded as 0-9; 1010 is decoded as a point; 1011 is decoded as minus (—); 1100, 1101, 1110, and 1111 are all decoded as blanks.

When a WRITE instruction is issued, the contents of the second output word are displayed in the numerical display, from left to right, starting with bits 0 through 3 for the leftmost digit. A character remains displayed on the numerical display until it is changed by a subsequent WRITE instruction. The numerical display is aligned digit by digit with the sixteen numerical switches. There is no direct physical or electrical connection, however.

The following list describes the code set used in decoding the information sent to the numerical display and in encoding the keys on the numerical switches.

BINARY CODE	CHARACTER DISPLAYED ON NUMERICAL DISPLAY	CORRESPONDING KEY ON NUMERICAL SWITCHES
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	. (decimal point)	- - -
1011	— (minus sign)	- - -
1100	Blank	- - -
1101	Blank	- - -
1110	Blank	- - -
1111	Blank	- - - *

* This code is generated when no digit key is depressed.

Binary Switches

The console has 32 binary toggle switches, arranged in two sets of 16 each and grouped by fours within each set. The switches have labels that can readily be changed. Each label is illuminated by the associated binary switch light when the corresponding switch is in the on position. When a READ instruction is issued, the settings of the binary switches make up the first four bytes of the third input word, from left to right, with bit 0 reflecting the state of the leftmost light. In contrast to the keys, the switches have only latching action but no neutral position.

Pluggable Entry Positions

To permit other switching and selection devices to be attached, there are 32 pluggable entry positions, each consisting of an input jack. Each pluggable entry corresponds to one of the binary switches described in the preceding section. When a plug is in the jack for any pluggable entry, the corresponding binary switch is ineffective.

When a pluggable entry is in use, it is represented by a binary zero if there is no connection between the terminals of its jack, and by a binary one if there is a closed connection. The bits from the pluggable entries substitute for the input bits from the corresponding switches; they constitute the first four bytes of the third input word.

There is a thirty-third jack that causes a channel signal to be sent to the computer whenever a connection is made between its terminals. This jack does not disable the signal key.

Binary Lights

For display of machine conditions, flags, indicators, and other binary information, there are 32 independent binary lights. They are located immediately behind the binary switches and are so grouped that there is visual correspondence between the binary lights and binary switches, even though there is no physical connection. Each of the sets of lights has a frame to hold a labeling strip.

When a WRITE instruction is issued, the contents of the first four bytes of the third output word are displayed in the binary lights, with those lights illuminated that receive binary ones. The lights remain on until turned off by another WRITE instruction.

Pluggable Exit Positions

In order to permit attachment of other display devices, the console has 32 pluggable exit positions, each consisting of an output jack. Each pluggable exit corresponds to one of the binary lights described in the previous section. Insertion of a plug in the jack does not disable the corresponding binary light.

When a pluggable exit is in use, it is active (provides current) when it is furnished a one-bit by means of a WRITE instruction. The exit remains active until another WRITE is given. Since the pluggable exits parallel the binary lights, the information addressed to the exits occupies the first four bytes of the third output word.

A thirty-third output jack furnishes a momentary current whenever the gong is sounded.

A thirty-fourth output jack parallels the wait light and provides a current when the wait light is on. The insertion of a plug in this jack does not disable the light.

Digital Potentiometers

To facilitate the entry of inherently analog information, there are three digital potentiometers. Each of these is a rotary switch with 128 positions encoded 0000000 to 1111111 in clockwise order. Each switch has a knob with a pointer that indicates the approximate setting of the potentiometer.

When a READ instruction is issued, the settings of the digital potentiometers constitute the rightmost seven bits of the fifth, sixth, and seventh bytes of the third input word. The leftmost bit in each byte is zero.

Keyboard

The keyboard is arranged in the usual fashion of a typewriter keyboard. There are 44 character keys and a shift key which indicates lower or upper case. As a character key is depressed, the associated character is automatically typed on the console printer. An 8-bit character code is associated with each character for entry to and exit from the computer. The space bar causes the printing mechanism to skip one printing position and enter a blank character consisting of all zeros.

Besides the character keys, the keyboard has several function keys. In addition to causing the indicated action on the printing mechanism, depression of each function key generates a corresponding function code for entry into the computer. The following are the function keys:

1. *Carriage Return.* The carriage return key causes the printing mechanism to advance to the first printing position on the next line.
2. *Backspace.* The backspace key causes the printing mechanism to return to the previous printing position on the same line.
3. *Tabulate.* The tabulate key causes the printing mechanism to skip to the next preset tabulate position to the right, on the same line.

When certain keys are held down, the action is repeated at top speed until the key is released. The following keys are provided with repeat action:

Space Bar
Carriage Return (only line feeding is repeated)
Backspace

All of the keys, including the character keys, are locked whenever the console is in the ready status but the keyboard is not active. The keyboard is active when the enter light is on, indicating that information can be entered. Manual operation is then possible until the end or erase key is depressed, which causes the keyboard to be locked again. The keyboard remains locked during a write operation. The keyboard is unlocked when the console is not ready so that the typewriter may be used for typing without entry to the computer.

If the shift lock is left in the upper shift position during manual entry, the action of locking the keyboard at the end of the operation automatically releases the shift lock. Thus, the keyboard is always in the lower shift after termination of manual entry.

Printer Mechanism Character and Function Codes

Each character of function code is sent from the keyboard to the exchange and from the exchange to the console printer as an 8-bit byte. Out of the total of 256 codes possible with eight bits, 175 distinct codes are assigned to the character codes that can be entered by means of the 44 character keys in conjunction with the upper and lower case shift key, and the ribbon shift key. One code corresponds to the space bar, and five codes are assigned to the control functions associated with the operation of the keyboard and the printing mechanism. The remaining codes are redundant and cannot be entered through the keyboard.

If the bits in each byte are numbered from 0 through 7, starting with the leftmost (most significant bit) a code for an upper or lower case character not modified by the depression of the ribbon shift key occupies the seven bit positions 1 through 7. The bit numbered 0 always has the value zero. Bit position 7 indicates case shift, with 0 representing lower case and 1 representing upper case. This set, consisting of 89 codes which correspond to 88 distinct non-blank characters and blank, is shown in the Appendix. The blank code, corresponding to the space bar on the keyboard, prints a blank space when sent to the console printer.

When a character is entered with the ribbon shift key depressed, the code corresponding to the character is modified to cause ribbon shift when the code is subsequently sent to the console printer for writing. This modification is limited to the three high-order bit positions of the byte, but the particular bits that are changed depend upon the character. Bit position 7 always indicates case shift, with 0 representing lower-case shift and 1 representing upper-case shift.

One ribbon-shift code corresponds to each of the 88 regular character codes. The ampersand (&) in conjunction with the ribbon shift, however, produces on input the same code as the space bar and, consequently, on output this code cannot be printed. Thus the ribbon-shift code set consists of 87 distinct codes that can be entered through the keyboard.

Figure 47 shows codes that are generated when a character is entered with the ribbon shift key depressed. When these codes are sent to the console printer during writing, the corresponding character is printed in the color associated with ribbon shift. It should be noted that because the ampersand in conjunction with ribbon shift produces on input the same code as the space bar, it cannot be printed on output.

Entry of Information

In order to enter information through the console, the console must have received a READ instruction from the computer. The operator can request such a READ instruction at any time by causing a channel-signal indication, provided the program so interprets the channel-signal indication.

BITS 4, 5, 6, 7	BITS 0, 1, 2, 3					
	0000	0001	1000	1001	1010	1011
0000	&	c	k	s	0	8
0001	+	C	K	S	*0*	*8*
0010	$	d	l	t	1	9
0011	=	D	L	T	*1*	*9*
0100	*	e	m	u	2	.
0101	(	E	M	U	*2*	:
0110	/	f	n	v	3	–
0111	)	F	N	V	*3*	?
1000	,	g	o	w	4	
1001	;	G	O	W	*4*	
1010	'	h	p	x	5	
1011	"	H	P	X	*5*	
1100	a	i	q	y	6	
1101	A	I	Q	Y	*6*	
1110	b	j	r	z	7	
1111	B	J	R	Z	*7*	

Figure 47. Ribbon Shift Codes, Console Printer and Keyboard

A channel-signal indication can be caused in two ways: by depressing the signal key or by depressing the enter key. The enter key, in addition to generating a channel signal, also enables the keyboard. The keyboard is disabled by the termination of the subsequent READ operation or by the completion of the first block of such operation if the multiple mode is used. If no data transfer follows the enabling, the keyboard is disabled by the next depression of the signal or end key.

When a READ instruction is received by the console with a keyboard not enabled, the block length is limited by the console to three words, and in the single-block mode the operation is automatically terminated when the three words scanned from the console switches have been sent to the exchange. When the multiple mode of reading is specified, the operation is repeated until the count in the control word is exhausted. As before, only the console switches are sensed and each block is limited to three words.

In order to use the keyboard for entry of information, the keyboard must have been enabled by depressing the enter key. The subsequent READ instruction, after scanning the three words from the console switches, then makes the keyboard active so that a message can be keyed. When the keyboard is active, the enter light is on and the keyboard is unlocked. As the message is keyed, a copy is always typed on the the console printer.

When the keyboard is enabled, a console block may be of any length; the length is not defined by the unit. To indicate the end of the message, the operator normally presses the end key. In the single mode this terminates the READ operation, disabling the keyboard. In the multiple mode, depression of the end key terminates the current block. However, since the termination of the block also disables the keyboard, the READ instruction subsequently is restricted to the console switches only and the block length is limited to three words. The scanning of the console switches then is repeated until the control word is exhausted. It is thus impossible to enter more than one message through the keyboard with a single READ instruction, even if the multiple mode is used. A depression of the end key does not interrupt the scanning of the console switches.

If the operator discovers an error while entering information through the keyboard, he may press the erase key. This terminates the READ operation and causes end-exception and end-of-operation indications unless the multiple-block flag in the control word is on. In the latter case only end-exception is given. The erase key is inoperative when the printing mechanism is not writing or when the keyboard is not active for entry of information.

Whenever a character or a keyboard function key is depressed, the corresponding 8-bit code is sent to the exchange and from there to core storage. Character and function codes may be intermixed in any order. If the shift key is manually locked in the upper position, the shift bit continues to be one. If the repeat action is used, the code is sent every time the action is repeated. When the character is entered with the ribbon shift key depressed, a modified character code is generated, as described before.

Both the end and erase keys generate a corresponding function code to be sent to the exchange before terminating the operation.

Writing under Computer Control

Information can be displayed on the console by means of a WRITE instruction. The first three words of the output message addressed to the console are interpreted by the binary lights and the numerical display. The remainder of the message, regardless of its length, is always used to actuate the console printer mechanism.

The block of information sent to the console can have any character and function codes in any order. The part of the message applying to the binary lights and the numerical display is decoded in accordance with the rules applicable to the respective display devices, as described above. All the subsequent bytes are interpreted by the console printer and are decoded either as control functions or characters. The three control functions carriage return, backspace, and tabulate initiate the same action as the corresponding keys, and the function codes are the same as those generated by these keys during manual entry. The keyboard is locked while printing under computer control is in progress.

The console printer message can consist of any number of lines of printing, but a carriage-return function code must be given on every line at or before the end of the printing line. Otherwise, overprinting of characters occurs at the extreme right. No indication concerning the end of the printing line is given to the computer. The maximum length of the printing line is 83 characters, but the actual length at any time depends upon the manual setting of margin controls.

When the number of character and function codes needed for one operation does not correspond to an integral number of core storage words, the last character sent to the console printer should be followed with an end code. In the single-block mode the end code causes the WRITE operation to be terminated with an end-of-operation regardless of whether or not the word count has reached zero. In the multiple-block mode printing on the console printer is terminated, but a new WRITE is immediately initiated at the console with the next full word in core storage as long as the count is not zero. From then on the operation proceeds as if a new WRITE had been issued, and after the first three words, which are interpreted by the binary lights and the numerical display, printing is resumed. In either case, the end code itself does not cause any printing or spacing. If the erase code is sent from the computer during writing, it is decoded as blank.

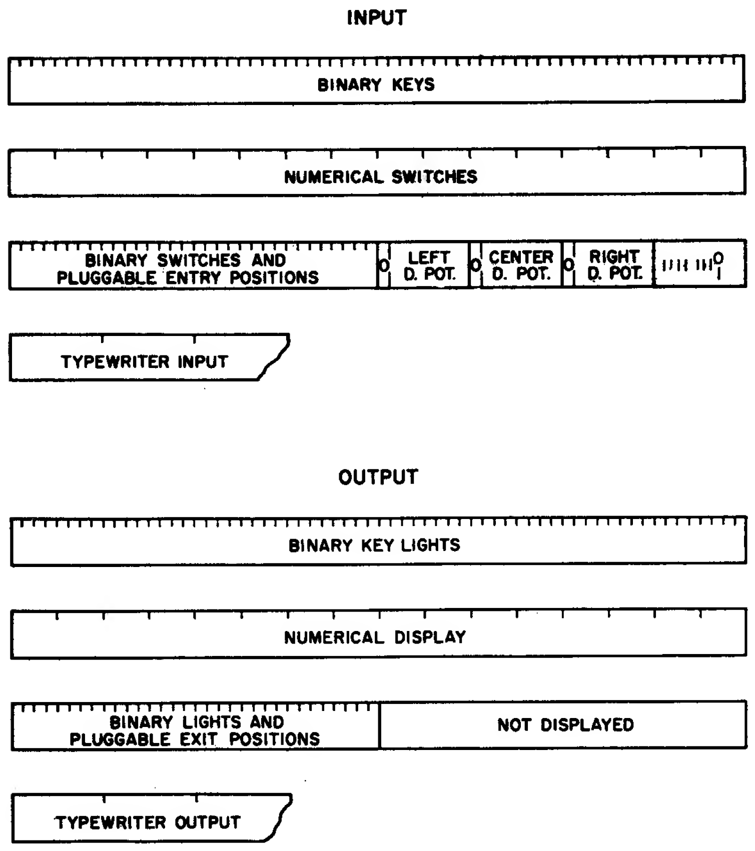


Figure 48. Console Message Formats

Information coming from the computer to the console is examined for function codes and forbidden characters only when it is directed to the console printer. All bit combinations can be used freely in the first three words of an output message.

Format

The allocation of bits in the first three input and output words is summarized in Figure 48. During input, the information scanned from the console switches occupies the whole three-word field except for the last byte. For the unused byte, the console provides all ones when the keyboard is enabled and the end code, bits 1111 1110 when it is disabled. During output, the binary lights and the numerical display are set up according to the contents of the first two and a half words. The second half of the third word is ignored. All words beyond the first three are entered from the keyboard or are printed on the console printer mechanism.

At the start of every WRITE operation, all the display positions and lights are turned off. Consequently, if a WRITE has fewer than three words, the display positions and lights corresponding to words not written remain turned off.

Redundant Character Codes

The bit combinations shown in Figure 49 are redundant character codes and cannot be generated by the keyboard during entry. When these bit combinations are sent to the console printer during writing, they initiate printing of the following characters. (RS means that the indicated character is printed with the ribbon shifted.)

BITS	BITS 0, 1, 2, 3					
4, 5, 6, 7	0111	1011	1100	1101	1110	1111
0000			k	s	K	S
0001			K	S	K	S
0010			l	t	l	t
0011			L	T	L	T
0100			m	u	m	u
0101			M	U	M	U
0110			n	v	n	v
0111			N	V	N	V
1000	&	&	o	w	o	w
1001	+	+	⊙	W	⊙	W
1010	$	$	p	x	p	x
1011	=	=	P	X	P	TAB
1100	*	*	q	y	q	BK SP
1101	(	(	Q	Y	Q	CR
1110	/	/	r	z	r	END
1111	)	)	R	Z	R	Z
		RS			RS	

Figure 49. Redundant Character Codes

High-Speed Disk System

The high-speed disk storage system provided with the IBM 7030 is composed of the IBM 7612 Disk Synchronizer, which provides the common control and data transfer paths for the disk system and the computer, and the storage unit, which provides large capacity auxiliary storage to supplement internal core storage.

IBM 7612 Disk Synchronizer

The IBM 7612 Disk Synchronizer serves as the major control for the disk system. It controls the execution of data transfer between a disk storage and core storage. Basically, the method of control is the same as that used in the exchange. However, the high rate of data transmission between a disk storage and internal storage causes some differences in operational and program control.

The action of the disk synchronizer (DS) may be summarized as follows: The central processing unit (CPU) initiates operation by sending an instruction to the DS. When the necessary interlock controls have been satisfied and the instruction accepted, control of the operation is transferred to the DS and the computer proceeds independently until the operation has been completed. All necessary control required for word assembly, word disassembly, and the disk file are provided by the DS, thus enabling it to operate independently of the CPU and exchange portions of the 7030 system. When the operation or operations specified have been completed, the CPU is signalled through the I-O status indicator bits of the interruption system.

The disk synchronizer has provision for addressing a maximum of 32 disk storage units. Each disk unit has its own channel address. However, only a single data transmission channel is available because of the high data transfer rate. Therefore, only one disk at a time may be used for reading or writing. Concurrently, however, other disks can be accepting and processing instructions for setting the read-write heads in position for subsequent data transmission.

Full words are transferred between the disk synchronizer and core storage. Each word consists of 64 information bits and eight error check and correction bits.

Each disk storage unit has a capacity of 2,097,152 (2^{21})words. Data may be transferred between the disk storage and core storage at an instantaneous rate of one full word every eight microseconds.

Each disk storage area is subdivided into 4096 (2^{12}) addressable locations called arcs, consecutively numbered from 0 through 4095. An arc contains 512 words of information and is the smallest addressable unit in the disk system. Each disk unit contains a total of 512 tracks, and each track contains eight arcs.

Physically, a single disk unit contains 39 disks, coated with a magnetic material. These are mounted on a common shaft, and are used for data storage. Other disks on the same shaft are used for address and arc control, timing and other control purposes. Each information disk has two surfaces that are used for data storage. The total of 78 disk faces is divided into two sets of 39 and each of these sets is equipped with 39 read-write heads. The 39 read-write heads are mounted on 20 access arms, which are on a common support. The set of heads and arms is called an access mechanism. All 39 heads move back and forth together during access motion to a track. One set of 39 faces contains all the odd-numbered tracks, the other set all even-numbered tracks. Thus, two consecutively numbered tracks are never located on the same set of disk faces. This arrangement makes it possible to position the heads of one access mechanism on the next track during the time the previous track is read or written.

Data on the disk storage are handled in bytes of 39 bits each. These bytes are read or written in parallel by the set of 39 read-write heads common to an access mechanism. Since a large number of bits are handled in parallel, it is practical to use error checking and correction (ECC) bits, and each 39 bit byte is composed of 32 data bits and seven ECC bits. The ECC bits accompany all data transferred to or from the high-speed disks, and, on reading, are used to correct a single bit error in a byte and detect double and most multiple errors in a byte. A 64-bit data word is divided into two bytes that are recorded on the disk storage in parallel columns 39 bits long and positioned in adjacent locations on the set of 39 disk faces.

Before transmission of data is started, it is necessary to set the access arms to the track containing the desired starting arc. After this initial positioning the arms will be automatically positioned as successive data transmission requires successive sequential arcs. Programmed repositioning is required only when non-sequential or random arcs are used.

Positioning of the arms under program control requires a maximum of 265 milliseconds (ms.) when the arms are moved from the outermost to the innermost track. Moving sequentially from one track to the next track, however, requires 51 ms. provided the tracks are in the same group of 16. Moving sequentially from the last track in one 16 group to the first track of the next 16 group, requires 117 ms.

After the access mechanism is first positioned, a rotational delay may be expected when a reading or writing operation is started. This may take a maximum of 34 ms. although an average of 17 ms. may be expected. If certain conditions of addressing and data block length are met, the rotational delay is reduced to a maximum of 4.2 ms. and an average of 2.1 ms. This feature is called "automatic access elimination."

While capacity of a disk unit is normally expressed in full words, 2,097,152 (2^{21}) words, the 7030 system uses binary-coded decimal (four bits per digit) and binary-coded alphameric (six bits per digit) data representations also. Therefore, the disk storage capacity can also be expressed as 134,217,728 (2^{27}) binary bits: 33,554,432 binary-coded decimal digits or 22,369,621 binary-coded alphameric characters.

A summary of the physical characteristics of the disk storage is:

Faces per disk storage	78
Tracks per set of 39 disk faces	256
Arcs per track	8
Data bits per byte	32
Bytes per word	2
Words per arc	512
Words per track	4096
Arcs per file	4096
Words per file	2,097,512 (2^{21})
Maximum track access	180 ms.
Maximum rotational access	33.3 ms.
Data transfer to storage (instantaneous rate)	8 microseconds/word

Data Paths

The Disk Synchronizer (DS) contains control features necessary to regulate the information flow over a single data channel linking core storage to disk storage. This is the read and write path for data to and from disk storage.

Since data transfers between the DS and core storage are on a full-word basis, the facility for word assembly and disassembly is provided in the Disk Synchronizer. Two words are buffered, one in a full-word register, the second in two half-word byte registers. Through the use of the three registers the data flow is synchronized between core storage and disk storage.

All word and byte transfers are checked for errors through the use of the error check and correction circuits associated with each type of register in the DS. These circuits use the error check and correction bits associated with the data bits during transfer.

Whenever data are assembled or disassembled, the associated ECC bits are not valid for the newly-formed word or byte. A new set of ECC bits must be generated to fit the new word or byte just formed. Through a mathematical relationship existing between old and new correction bits, the continuity of checking is preserved by tests made in the DS before further processing of the data.

Data Transmission

The disk system and the exchange differ in their operation in the following ways:

1. Only one disk storage at a time may be reading or writing.
2. No use is made of control word flag bits or refill field and they are discarded after checking the control word error correction and check bits. Only the data word address field and word count field of the control word are retained and used by the disk synchronizer.
3. In executing COPY CONTROL WORD, only the data word address and word count fields will be returned to core storage. COPY CONTROL WORD is executed only if the system is not reading or writing.

Indicators

The indicators and conditions which turn them on are reviewed in this section, with emphasis on conditions resulting from operation of the disk system.

General Indicators

Address Invalid (AD). The purpose of this indicator, when on in conjunction with input-output instructions, signals that the right effective address of a READ, WRITE, or COPY CONTROL WORD instruction is 0 through 31. When an error is found which turns on this indicator, the processing of the instruction is terminated in the CPU.

Input-Output Reject Indicators

Exchange Check Reject (EKJ). This indicator is turned on when an error is detected by the disk synchronizer or disk storage in the course of setting up the present instruction. The error detected may be caused by:

1. Parity errors in the information contained in

2. Specification of disk storage or channel addresses that are not available to the programmer. This includes uninstalled channels or channels that are not operative because no disk storage has been provided for them. When a disk storage is temporarily physically disconnected from the disk synchronizer, the channel will be in a not-ready status and the disk synchronizer will not set the exchange-check-reject indicator.
3. An attempt is made to locate to or process data to or from a disk arc with an arc address greater than 4095.

Unit Not Ready Reject (UNRJ). This indicator is turned on when an instruction is given to a disk storage that is not ready to be operated. This is usually caused when a disk storage is temporarily disconnected from its channel.

Channel Busy Reject (CBJ). Because of the restrictions on addressing and simultaneous operation of disk storage, this indicator can be turned on by a number of conditions that do not exist in the exchange. The conditions which can be responsible for channel-busy-reject in the disk synchronizer are:

1. A READ, WRITE or LOCATE instruction is addressed to a disk which is executing a previous instruction, or is waiting to make a program interruption.
2. A READ or WRITE instruction is addressed to a disk storage at a time when some other disk storage is engaged in a reading or writing operation.
3. A COPY CONTROL WORD instruction is issued at a time when a disk unit is engaged in reading or writing.

INPUT-OUTPUT STATUS INDICATORS

When an operation associated with a disk storage has been terminated, the status at that time is recorded. Each channel will retain the status bits associated with termination of the last instruction executed or attempted by the channel or disk synchronizer until the priority sequencing controls permit it to interrupt in the central processing unit.

Status information can occur prior to, during or subsequent to data transmission. Certain checks that occur while data transmission is in progress are held up in the disk synchronizer or disk storage until the end-of-operation is signalled. Other checks can cause immediate termination of the instruction being executed.

Exchange Program Check (EPGK). This indicator is turned on by a program error. When turned on it usually implies that the operation cannot be completed. Conditions which set EPGK are:

1. A CONTROL has been issued to a disk storage.
2. The DS attempts to transfer a word to or from a core storage location to which it does not have access (locations 0 through 31 or in locations above the maximum address available in core storage). In the case of a WRITE instruction to the disk unit with a data word address too high, the DS writes zeros in the arc specified in the locate register. The EPGK indicator is delayed until the write zeros operation is completed.
3. A nonexistent disk address is encountered by attempting to read or write beyond arc 4095.

Programming errors indicated above are detected after the instruction is sent to the disk system. The EPGK indicator is set soon after the instruction has been accepted by the disk synchronizer. However, by the time an interruption can be made, the computer has proceeded with its instruction sequence. The specified disk storage will not perform the instruction.

The EPGK indicator is set, by invalid address detection, when an attempt is made to transfer a word to or from the location specified by the invalid address. The operation proceeds normally until the invalid address is detected.

Unit Check (UK). This indicator is turned on when an uncorrectable data error or machine malfunctioning occurs. Two classes of errors that will turn on UK are distinguished by the disk system:

1. Those errors which cause immediate termination of the instruction and immediate interruption. These errors are developed by malfunctioning of the disk storage and the disk synchronizer.
2. Those errors which permit the operation to be completed. These errors include uncorrectable data and addressing parity errors. This class of errors will set the UK and EOP indicator bits.

End Of Operation (EOP). When this indicator is turned on and no other check interruption occurs, it indicates that the instruction has been completed successfully. It will be set for all instructions except COPY CONTROL WORD and those instructions specifying SEOP. In this latter case, however, the suppression is ignored if a unit-check is also turned on and the operation is completed.

Keys and Lights

A disk file switch turns the 60-cycle power on and off. The 400 cycle power for the disk units is turned on and off together with power for the disk synchronizer.

The disk units do not require operator access, and can be installed outside of the operating area.

Operations

The input-output instructions used for the operation of the disk system are described here, with emphasis on characteristics that are different from use of the same instructions with the exchange. They are summarized below:

	OPERATION	ACTION	CONTROL WORD
RD	Read (SEOP)	Data from disk storage to core storage	yes
W	Write (SEOP)	Data from core storage to disk storage	yes
LOC	Locate (SEOP)	Set access mechanism to initial arc for later data transmission	no
REL	Release (SEOP)	Terminate data transmission immediately	no
CCW	Copy Control Word	Data word address field and word count field of control word to core storage	no

When the SEOP modification is used it means SUPPRESS END OF OPERATIONS. The normal EOP indication given at the successful conclusion of an operation will be suppressed. The CONTROL and CONTROL (SEOP) instructions have no meaning for disk storage and will set the EPGK indicator.

Read (RD); Read SEOP (RDSEOP)

These instructions initiate a reading operation from the disk specified by the channel address, bit positions 12 through 18, in the left effective address of these instructions. The right effective address of the instruction, bit positions 32 through 49, specifies the core storage location of the control word associated with these operations.

Data are transferred to successive locations in core storage beginning with the location specified in the data word address field of the control word. The number of words of data read and stored is given by the value in the count field of the control word.

Used with disk storage, these instructions differ from the operations as used with the exchange in that the control word flag bits and refill fields are ignored. Data will be transmitted from successive arcs of the disk until the count field is reduced to zero. However, the data read need not be a multiple of the number of words contained in an arc.

READ SEOP differs from READ in that the EOP indication will be suppressed when the operation is completed normally. Otherwise, the operations are the same.

INDICATORS

Exchange Check Reject (EKJ)
Channel Busy Reject (CBJ)
Unit Not Ready Reject (UNRJ)
Exchange Program Check (EPGK)
Unit Check (UK)
End of Operation (EOP)
Address Invalid (AD)

Write (W); Write SEOP (WSEOP)

These instructions initiate a writing operation to the disk specified by the channel address, bit positions 12 through 18, in the left effective address of these instructions. The right effective address of the instruction, bit positions 32 through 49, specifies the core storage location of the control word associated with these operations.

Data are transferred from successive locations in core storage beginning with the location specified in the data word address field of the control word. The number of words of data written is given by the value in the count field of the control word.

Used with a disk storage, these instructions differ from the operations as used with the exchange in that the control word flag bits and refill field are ignored. Data will be written into successive arcs of the disk until the count field is reduced to zero. The data written need not be a multiple of the number of words contained in an arc. In this case, however, the remainder of the last arc will be automatically filled with zeros.

WRITE SEOP differs from WRITE in that the EOP indication will be suppressed when the operation is completed normally. Otherwise they are the same.

INDICATORS

Exchange Check Reject (EKJ)
Channel Busy Reject (CBJ)
Unit Not Ready Reject (UNRJ)
Exchange Program Check (EPGK)
Unit Check (UK)
End Of Operation (EOP)
Address Invalid (AD)

Locate (LOC); Locate SEOP (LOCSEOP)

These operations are used to initiate setting of the access mechanism to the track which contains the specified arc. The disk is specified by the channel address, bit positions 12 through 18, in the left effective address of these instructions. The right effective address, bit positions 32 through 49, of the instruction specifies the arc address. Positions 32-37 must be 0 bits while positions 38-49 specify the arc address.

Specification of the arc automatically identifies the access mechanism to be used on that arc. These instructions cause the proper access mechanism of the pair attached to each disk storage to be positioned at the track containing the specified arc. Concurrently, the other access mechanism is positioned on the track immediately following the specified track. When both access mechanisms have been positioned, an EOP indication is given. The starting arc within the track is retained within the disk unit.

A LOCATE or LOCATE SEOP may be given to any disk attached to the data synchronizer at any time if the disk specified is not busy with data transmission or a prior LOCATE.

These operations specify an initial arc for which data transmission may occur. Because of these instructions, the disk units are randomly addressable by arc. Whenever successive arcs are to be read or written, only the initial arc need be specified.

LOCATE SEOP differs from LOCATE in that th : EOP indication is suppressed when the operation is completed normally. Otherwise they are the same.

INDICATORS

Exchange Check Reject (EKJ)
Channel Busy Reject (CBJ)
Unit Not Ready Reject (UNRJ)
Exchange Program Check (EPGK)
Unit Check (UK)
End of Operation (EOP)

PROGRAMMING NOTE

The arc address specified in the LOCATE instructions may be considered as having two parts:

1. Track address: high-order nine bits, bit positions 38 through 46.
2. Arc within track address: low-order three bits, bit positions 47 through 49.

The LOCATE operations position the read-write heads of the access mechanism to the track specified in the arc address. Subsequent READ or WRITE operations will then cause the arc within the track to be found and data transmission will begin. Associated with each disk is an arc address register. As successive arcs are used, this register is increased by one for each arc. When required, the access mechanism will be automatically repositioned to the next higher track. Thus the alternating access mechanisms may provide continuous data flow.

Release (REL); Release SEOP (RELSEOP)

These instructions immediately terminate any data transmission operation currently in process, at the disk unit specified by the channel address, bit positions 12 through 18 of the left effective address.

When a disk unit is released, the arc address register is left at the address of the last arc used unless transmission of that arc has been completed. If the disk unit was writing, the remainder of the arc, in which the RELEASE operation was accepted, will have zeros recorded. A LOCATE operation will proceed to its normal end and the arc address register will be set as specified.

If the disk unit is not ready, the disk system will not accept the instruction. Instead, the unit-not-ready-reject indicator is turned on. In this respect these operations are different from the exchange.

RELEASE SEOP differs from RELEASE in that the EOP indication will be suppressed when the operation is completed normally. This instruction resets all status indications during its execution.

INDICATORS

Unit Not Ready Reject (UNRJ)
End of Operation (EOP)

Copy Control Word (CCW)

The left effective address, bit positions 12 through 18, of the instruction specify the channel address. The right effective address, bit positions 32 through 49, of the instruction specify the core storage location to which the information is to be sent.

The disk synchronizer does not have control word storage as does the exchange. The data word address and word count fields are retained in registers in the disk synchronizer and are used to control core storage references. Since only one disk unit may transmit at a time, these registers are associated with the last channel which was performing a data transmission operation. The contents of these registers are placed in a full-word format, corresponding to their location in a control word. Since the execution of a COPY CONTROL WORD instruction uses the transmission channel of the disk synchronizer, the instruction is not executed when a disk storage is reading or writing, and the channel-busy indicator will be turned on.

The normal completion of this instruction will not give an EOP indication, since when this instruction is used with the disk system it must be completed before the computer proceeds with the next instruction.

INDICATORS

Exchange Check Reject (EKJ)
Channel Busy Reject (CBJ)
Unit Not Ready Reject (UNRJ)
Address Invalid (AD)

Automatic Access Elimination

This feature of the disk system provides an automatic method for minimizing rotational access time when full tracks of information are to be read or written. On the average the effective access time is reduced by a factor of eight.

The disk synchronizer automatically tests for three conditions on each read or write operation. If all three conditions are satisfied, data will automatically be handled in the manner described below. The conditions are:

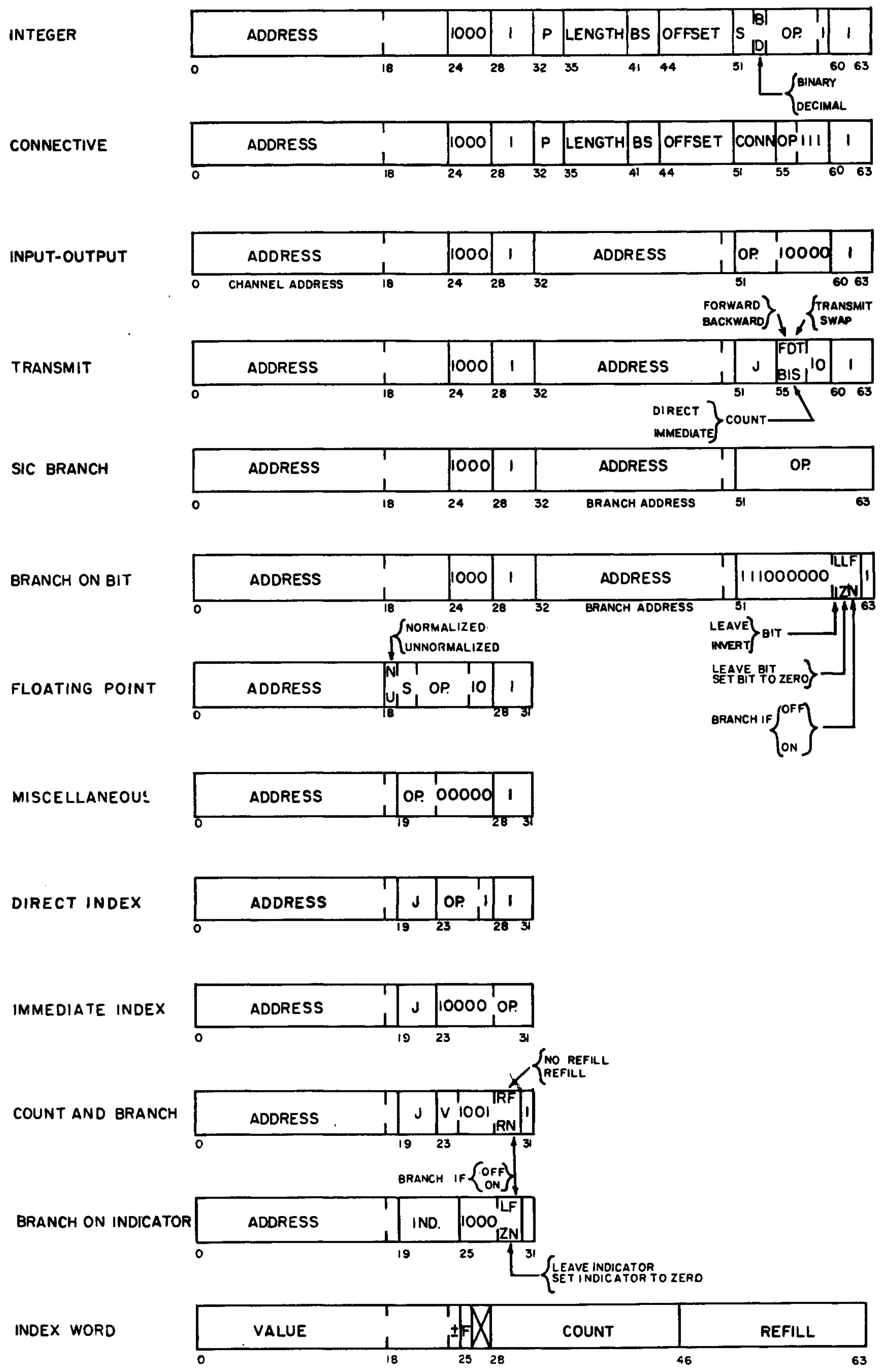
1. The arc address at the disk file must have the three low-order bit positions equal to zero. That is, the arc address must specify arc 0 of this track.
2. The twelve low-order bits of the data word address field of the control word must be zero. Thus the data read or written will be stored or obtained from core storage locations starting with address 4096 or a multiple of 4096.
3. The word count field of the control word must be 4096. Therefore, only one track of information can be transferred with automatic access elimination.

If these three conditions are satisfied, the disk begins transmission operations at the start of the next arc it senses. The data word address is automatically adjusted to correspond to this arc's data address. Transmission continues from this point until transmission associated with arc 7 of this track has been completed. The data word address is then automatically adjusted to the initial data word address and transmission continues with arc 0 until the starting point is reached, i.e., 4096 words have been transmitted.

When successive tracks are to be read or written an automatic adjustment of the arc address is made to its next track and transmission continues. At the completion of data transmission the arc address register will be left at arc zero (0) of the next sequential track if the operation was terminated at sector 8.

The data word address register contains the core storage location of the last word read, which depends on the sector in which the read operation started.

Instruction Formats


INTEGER
ADDRESS | 1000 | I | P | LENGTH | BS | OFFSET | S | B/D | OP. | I | I
0 18 24 28 32 35 41 44 51 60 63
BINARY
DECIMAL
CONNECTIVE
ADDRESS | 1000 | I | P | LENGTH | BS | OFFSET | CONN | OP 111 | I
0 18 24 28 32 35 41 44 51 55 60 63
INPUT-OUTPUT
ADDRESS | 1000 | I | ADDRESS | OP. | 10000 | I
0 CHANNEL ADDRESS 18 24 28 32 51 60 63
FORWARD
BACKWARD
TRANSMIT
SWAP
TRANSMIT
ADDRESS | 1000 | I | ADDRESS | J | FDT BIS | 10 | I
0 18 24 28 32 51 55 60 63
DIRECT
IMMEDIATE
COUNT
SIC BRANCH
ADDRESS | 1000 | I | ADDRESS | OP.
0 18 24 28 32 BRANCH ADDRESS 51 63
BRANCH ON BIT
ADDRESS | 1000 | I | ADDRESS | 111000000 | LLF ZN | I
0 18 24 28 32 BRANCH ADDRESS 51 63
LEAVE
INVERT
BIT
LEAVE BIT
SET BIT TO ZERO
BRANCH IF OFF ON
NORMALIZED
UNNORMALIZED
FLOATING POINT
ADDRESS | N U | S | OP. | 10 | I
0 18 28 31
MISCELLANEOUS
ADDRESS | OP. | 00000 | I
0 19 28 31
DIRECT INDEX
ADDRESS | J | OP. | I | I
0 19 23 28 31
IMMEDIATE INDEX
ADDRESS | J | 10000 | OP.
0 19 23 31
NO REFILL
REFILL
COUNT AND BRANCH
ADDRESS | J | V | 1001 | RF RN | I
0 19 23 31
BRANCH IF OFF ON
BRANCH ON INDICATOR
ADDRESS | IND. | 1000 | LF ZN
0 19 25 31
LEAVE INDICATOR
SET INDICATOR TO ZERO
INDEX WORD
VALUE | ±F | COUNT | REFILL
0 18 25 28 46 63

List of Indicators

No.	Mne-monic	Mask	Class	Name
Equipment Check				
0	MK	1	P,H	Machine Check
1	IK	1	P,H	Instruction Check
2	IJ	1	P,S	Instruction Reject
3	EK	1	P,C	Exchange Control Check
Attention Request				
4	TS	1	P,C	Time Signal
5	CPUS	1	P,C	CPU Signal
Input-Output Rejects				
6	EKJ	1	P,S	Exchange Check Reject
7	UNRJ	1	P,S	Unit Not Ready Reject
8	CBJ	1	P,S	Channel Busy Reject
Input-Output Status				
9	EPGK	1	P,C	Exchange Program Check
10	UK	1	P,C	Unit Check
11	EE	1	P,C	End Exception
12	EOP	1	P,C	End of Operation
13	CS	1	P,C	Channel Signal
14		1		Reserved
Instruction Exception				
15	OP	1	P,S	Operation Code Invalid
16	AD	1	P,S	Address Invalid
17	USA	1	P,S	Unended Sequence of Addresses
18	EXE	1	P,S	Execute Exception
19	DS	1	P,S	Data Store
20	DF	m	P,S*,C	Data Fetch
21	IF	m	P,S*,C	Instruction Fetch

*Class S when mask is one; otherwise Class C.

No.	Mne-monic	Mask	Class	Name
Result Exception				
22	LC	m	P,C	Lost Carry
23	PF	m	P,C	Partial Field
24	ZD	m	P,C	Zero Divisor
Result Exception - Floating Point				
25	IR	m	P,C	Imaginary Root
26	LS	m	P,C	Lost Significance
27	PSH	m	P,C	Preparatory Shift Greater than 48
28	XPFP	m	P,C	Exponent Flag Positive: $\text{Exp.} \geq 2^{10}$
29	XPO	m	P,C	Exponent Overflow: $\text{Exp.} \geq 2^{10}$
30	XPH	m	P,C	Exponent High: $2^{10} > \text{Exp.} \geq 2^{9}$
31	XPL	m	P,C	Exponent Low: $2^{9} > \text{Exp.} \geq 2^{6}$
32	XPU	m	P,C	Exponent Underflow: $\text{Exp.} \leq -2^{10}$
33	ZM	m	T,C	Zero Multiply
34	RU	m	P,C	Remainder Underflow

No.	Mne-monic	Mask	Class	Name	
Flagging					
35	TF	m	T,C	Data Flag T	Reset by variable field length and floating point operations requiring data fetch.
36	UF	m	T,C	Data Flag U	
37	VF	m	T,C	Data Flag V	
38	XF	m	T,S*,C	Index Flag - Reset by index arithmetic operations	

*Class S only during index branching when the mask is one and the system enabled; otherwise Class C.

No.	Mne-monic	Mask	Class	Name	
Transit Operations					
39	BTR	m	P,C	Binary Transit	
40	DTR	m	P,C	Decimal Transit	
Program					
41- 47	PG0- PG6	m	P,C	Program Indicators Zero through Six	
48	XCZ	0	T,C	Index Count Zero	Reset by index arithmetic ops. other than comparisons
49	XVLZ	0	T,C	Index Value Less than Zero	
50	XVZ	0	T,C	Index Value Zero	
51	XVGZ	0	T,C	Index Value Greater than Zero	
52	XL	0	T,C	Index Low	Reset by index comparison operations.
53	XE	0	T,C	Index Equal	
54	XH	0	T,C	Index High	
Arithmetic Result					
55	MOP	0	T,C	To-Memory Op.	Reset by variable field length or floating point op.
56	RLZ	0	T,C	Result Less than Zero	Reset by variable field length or floating point ops. except comparisons
57	RZ	0	T,C	Result Zero	
58	RGZ	0	T,C	Result Greater than Zero	
59	RN	0	T,C	Result Negative	
60	AL	0	T,C	Accum. Low	Reset by comparison operations
61	AE	0	T,C	Accum. Equal	
62	AH	0	T,C	Accum. High	
Mode					
63	NM	0	P,C	Noisy Mode	

T	Indicator temporary and is reset by later operations.
P	Indicator permenent but may be turned off by interruption or during BI.
C	The execution of the instruction during which the indicator is actuated is completed.
H	The execution of the instruction during which the indicator is actuated is terminated.
S	The execution of the instruction during which the indicator is actuated is suppressed, except during EX,EXIC,T, and SWAP.

Storage Assignment

Location		Name	Length	Bit Address
0	ext	Zero	64	0-63
1	ind	Interval Timer (P,a)	19	0-18
1	ind	Time Clock (P,b)	36	28-63
2	ext	Interruption Address (P)	18	0-17
3	int	Upper Boundary (P)	18	0-17
3	int	Lower Boundary (P)	18	32-49
3	int	Boundary Control Bit (P)	1	57
4	ext	Maintenance Bits	64	0-63
5	int	Channel Address (b)	7	12-18
6	int	Other CPU	19	0-18
7	int	Left Zeros Count	7	17-23
7	int	All Ones Count	7	44-50
8	int	Left Half of Accumulator	64	0-63
9	int	Right Half of Accumulator	64	0-63
10	int	Accumulator Sign	8	0-7
11	int	Indicators (c)	64	0-63
12	int	Mask (d)	64	0-63
13	ext	Remainder	64	0-63
14	ext	Factor	64	0-63
15	ext	Transit	64	0-63
16-31	ind	Index Registers X0-X15	64	0-63

P	=	Permanently protected area of storage
a	=	Read-only except for Store Value, Store Count, Store Refill, and Store Address
b	=	Read-only
c	=	Bit positions 0-19 are read-only
d	=	Bit positions 0-19 are always ones and positions 48-63 are always zeros
ext	=	External storage location
ind	=	Index core storage location
int	=	Internal registers

Character Codes

BITS 4-5-6-7	BITS 0-1-2-3							
	0000	0001	0010	0011	0100	0101	0110	0111
0000	BLANK		&				0	8
0001				C	K	S		
0010			$				1	9
0011				D	L	T		
0100			*				2	.
0101				E	M	U		
0110			/				3	–
0111				F	N	V		
1000		%	,				4	
1001				G	O	W		
1010		◊	'				5	
1011				H	P	X		
1100		#					6	
1101			A	I	Q	Y		
1110		@					7	
1111			B	J	R	Z		

49 Character Set

BITS 4-5-6-7	BITS 0-1-2-3							
	0000	0001	0010	0011	0100	0101	0110	0111
0000	BLANK		&	c	k	s	0	8
0001			+	C	K	S	$_{0}$	$_{8}$
0010			$	d	l	t	1	9
0011			=	D	L	T	$_{1}$	$_{9}$
0100			*	e	m	u	2	.
0101			(	E	M	U	$_{2}$	:
0110			/	f	n	v	3	–
0111			)	F	N	V	$_{3}$	?
1000			,	g	o	w	4	
1001			;	G	O	W	$_{4}$	
1010			'	h	p	x	5	
1011			"	H	P	X	$_{5}$	
1100			a	i	q	y	6	
1101			A	I	Q	Y	$_{6}$	
1110			b	j	r	z	7	
1111			B	J	R	Z	$_{7}$	

89 Character Set

BITS 4-5-6-7	BITS 0-1-2-3							
	0000	0001	0010	0011	0100	0101	0110	0111
0000	BLANK		–	+				
0001	1	/	J	A				
0010	2	S	K	B				
0011	3	T	L	C				
0100	4	U	M	D				
0101	5	V	N	E				
0110	6	W	O	F				
0111	7	X	P	G				
1000	8	Y	Q	H				
1001	9	Z	R	I				
1010	0	:						
1011	=	,	$	.				
1100	'	(	*	)				
1101								
1110								
1111								

48 Character Set

BITS 4-5-6-7	BITS 0-1-2-3							
	0000	0001	0010	0011	0100	0101	0110	0111
0000	BLANK	[	&	c	k	s	0	8
0001	±	⊃	+	C	K	S	$_{0}$	$_{8}$
0010	→	]	$	d	l	t	1	9
0011	≠	°	=	D	L	T	$_{1}$	$_{9}$
0100	∧	←	*	e	m	u	2	.
0101	{	≡	(	E	M	U	$_{2}$	:
0110	↑	¬	/	f	n	v	3	–
0111	}	√	)	F	N	V	$_{3}$	?
1000	∨	%	,	g	o	w	4	
1001	∀	\	;	G	O	W	$_{4}$	
1010	↓	◊	'	h	p	x	5	
1011	‖	|	"	H	P	X	$_{5}$	
1100	>	#	a	i	q	y	6	
1101	≧	!	A	I	Q	Y	$_{6}$	
1110	<	@	b	j	r	z	7	
1111	≦	~	B	J	R	Z	$_{7}$	

120 Character Set

Alphabetical Listing of Operation Codes

Mnemonic	Name
+	Add
+	Add
+MG	Add to Magnitude
+MG	Add to Magnitude
-	Subtract
-	Subtract
-MG	Subtract from Magnitude
-MG	Subtract from Magnitude
*	Multiply
*	Multiply
*+	Multiply and Add
*+	Multiply and Add
*A +	Multiply Absolute and Add
*I +	Multiply Immediate and Add
*N +	Multiply Negative and Add
*N +	Multiply Negative and Add
*NA +	Multiply Negative Absolute and Add
*NI +	Multiply Negative Immediate and Add
/	Divide
/	Divide
B	Branch
BB	Branch on Bit
BB1	Branch on Bit and Set to One
BBN	Branch on Bit and Negate
BBZ	Branch on Bit and Zero
BD	Branch Disabled
BE	Branch Enabled
BEW	Branch Enabled and Wait
BR	Branch Relative
BZB	Branch on Zero Bit
BZB1	Branch on Zero Bit and Set to One
BZBN	Branch on Zero Bit and Negate
BZBZ	Branch on Zero Bit and Zero
C	Connect
C + 1	Add Immediate to Count
C - 1	Subtract Immediate from Count
CB	Count and Branch
CBR	Count, Branch, and Refill
CBZ	Count and Branch on Zero Count
CBZR	Count, Branch on Zero Count, and Refill
CCW	Copy Control Word
CM	Connect to Memory
CT	Connect for Test
CTL	Control
CV	Convert
D +	Add Double
D + MG	Add Double to Magnitude
D -	Subtract Double
D - MG	Subtract Double from Magnitude
DCV	Convert Double
DL	Load Double
DLWF	Load Double with Flag
D*	Multiply Double
D/	Divide Double
E +	Add to Exponent
E + AI	Add Absolute Immediate to Exponent
E + I	Add Immediate to Exponent
E -	Subtract From Exponent
E - AI	Subtract Absolute Immediate from Exponent
E - I	Subtract Immediate from Exponent
EX	Execute
EXIC	Execute Indirect and Count
F +	Add to Fraction
F -	Subtract from Fraction
K	Compare
K	Compare
KC	Compare Count
KCI	Compare Count Immediate
KE	Compare If Equal
KF	Compare Field
KFE	Compare Field If Equal
KFR	Compare Field for Range
KLN	Check Light On
KMG	Compare Magnitude
KMGR	Compare Magnitude for Range
KR	Compare for Range
KR	Compare for Range
KV	Compare Value
KVI	Compare Value Immediate
KVNI	Compare Value Negative Immediate
L	Load
L	Load
LC	Load Count
LCI	Load Count Immediate
LCV	Load Converted
LF	Load Field
LFT	Load Factor
LFT	Load Factor
LOC	Locate (same as Select Unit)
LR	Load Refill
LRI	Load Refill Immediate
LV	Load Value
LVE	Load Value Effective
LVI	Load Value Immediate
LVNI	Load Value Negative Immediate
LVS	Load Value with Sum
LX	Load Index
LTRCV	Load Transit Converted
LTRS	Load Transit and Set
LWF	Load with Flag
LWF	Load with Flag
M +	Add to Memory
M +	Add to Memory
M + 1	Add One to Memory
M + A	Add to Absolute Memory
M + MG	Add Magnitude to Memory
M + MG	Add Magnitude to Memory
M -	Subtract from Memory
M -	Subtract from Memory
M -1	Subtract One from Memory
M -A	Subtract from Absolute Memory
M -MG	Subtract Magnitude from Memory
M -MG	Subtract Magnitude from Memory
NOP	No Operation
R	Refill
RCZ	Refill on Count Zero
RD	Read
REL	Release
REW	Rewind
RNX	Rename
R/	Reciprocal Divide
SC	Store Count
SEOP	Suppress End of Operation
SF	Store Field
SHF	Shift Fraction
SHFL	Shift Fraction Left (same as SHFA)
SHFR	Shift Fraction Right (same as SHFNA)
SIC	Store Instruction Counter If
SLO	Store Low Order
SNRT	Store Negative Root
SR	Store Refill
SRD	Store Rounded
SRD	Store Rounded
SRT	Store Root
ST	Store
ST	Store
SU	Select Unit (same as Locate)
SV	Store Value
SVA	Store Value in Address
SWAP	Swap
SWAPI	Swap Immediate
SWAPB	Swap Backward
SWAPBI	Swap Backward Immediate
SX	Store Index
T	Transmit
TI	Transmit Immediate
TB	Transmit Backward
TBI	Transmit Backward Immediate
V +	Add to Value
V + I	Add Immediate to Value
V + C	Add to Value and Count
V + CR	Add to Value, Count, and Refill
V + IC	Add Immediate to Value and Count
V + ICR	Add Immediate to Value, Count, and Refill
V - I	Subtract Immediate from Value
V - IC	Subtract Immediate from Value and Count
V - ICR	Subtract Immediate From Value, Count, and Refill
W	Write
WEF	Write End-of-File
Z	Store Zero

Complete Operation and Modifier List

INTEGER ARITHMETIC OPERATIONS; Instruction bits 51-63

ssbxxxxx1 iiii	Mnemonic	Name	Page
Operations			
00000	+	Add	60
10000	+MG	Add to magnitude	60
00010	L	Load	61
10010	LWF	Load with flag	61
00100	M+	Add to memory	61
10100	M+MG	Add magnitude to memory	62
10101	M+1	Add one to memory	62
00110	ST	Store	63
10110	SRD	Store rounded	64
01000	K	Compare	65
11000	KF	Compare field	65
01001	KE	Compare if equal	65
11001	KFE	Compare field if equal	66
01010	KR	Compare for range	66
11010	KFR	Compare field for range	67
11110	LTRS	Load transit and set	67
01100	*	Multiply	68
00101	LFT	Load factor	69
11100	*+	Multiply and add	69
01110	/	Divide	70
Classes			
0		Binary	
1		Decimal	
Modifiers			
0		Signed	
1		Unsigned	
0		Same sign	
1		Negative sign	

RADIX CONVERSION OPERATIONS; Instruction bits 51-63

ssbxxxxx1iiii	Mnemonic	Name	Page
Operations			
00001	LCV	Load converted	72
10001	LTRCV	Load transit converted	73
01101	CV	Convert	73
11101	DCV	Convert double	74
Classes			
0		Binary	
1		Decimal	
Modifiers			
0		Signed	
1		Unsigned	
0		Same sign	
1		Negative sign	

CONNECTIVE OPERATIONS; Instruction bits 51-63

ccccxx111iiii	Mnemonic	Name	Page
Operations			
00	C	Connect	76
01	CM	Connect to memory	77
10	CT	Connect for test	77

FLOATING POINT OPERATIONS; Instruction bits 18-31

nssxxxxx10iiii	Mnemonic	Name	Page
Operations			
00000	+	Add	86
01000	+MG	Add to magnitude	88
00001	L	Load	88
01001	LWF	Load with flag	88
00010	M+	Add to memory	88
01010	M+MG	Add magnitude to memory	89
00011	ST	Store	90
00100	K	Compare	90
01100	KMG	Compare magnitude	91
00101	KR	Compare for range	91
01101	KMGR	Compare magnitude for range	92
00110	*	Multiply	92
00111	/	Divide	93
01111	R/	Reciprocal divide	94
11011	SRT	Store root	94
10000	D+	Add double	95
11000	D+MG	Add double to magnitude	96
10001	DL	Load double	96
11001	DLWF	Load double with flag	96
01011	SRD	Store rounded	96
10011	SLO	Store low order	97
10110	D*	Multiply double	97
10010	LFT	Load factor	98
01110	*+	Multiply and add	98
10111	D/	Divide double	99
10100	F+	Add to fraction	101
11100	SHF	Shift fraction	102
10101	E+	Add to exponent	102
11101	E+I	Add immediate to exponent	102
Classes			
0		Normalized	
1		Unnormalized	
Modifiers			
0		Same sign	
1		Absolute sign	
0		Same sign	
1		Negative sign	

DIRECT INDEX ARITHMETIC; Instruction bits 19-31

iiiixxxx1iiii	Mnemonic	Name	Page
Operations			
0000	LX	Load index	23
0001	LV	Load value	23
0010	LC	Load count	24
0011	LR	Load refill	24
1000	SX	Store index	24
1001	SV	Store value	24
1010	SC	Store count	24
1011	SR	Store refill	24
0101	V+	Add to value	24
0110	V+C	Add to value and count	24
0111	V+CR	Add to value, count, and refill	24
0100	KV	Compare value	24
1100	KC	Compare count	24
1111	RNX	Rename	27
1101	LVE	Load value effective	28
1110	SVA	Store value in address	29

Complete Operation and Modifier List

IMMEDIATE INDEX ARITHMETIC; Instruction bits 19-31

jjjj10000xxxx	Mnemonic	Name	Page
Operations			
0001	LVI	Load value immediate	25
0010	LCI	Load count immediate	25
0011	LRI	Load refill immediate	25
1001	LVNI	Load value negative immediate	25
0101	V+I	Add immediate to value	25
0110	V+IC	Add immediate to value and count	25
0111	V+ICR	Add immediate to value, count and refill	25
1101	V-I	Subtract immediate from value	25
1110	V-IC	Subtract immediate from value and count	25
1111	V-ICR	Subtract immediate from value, count and refill	25
0000	C+I	Add immediate to count	25
1000	C-I	Subtract immediate from count	25
0100	KVI	Compare value immediate	25
1100	KVNI	Compare value negative immediate	25
1010	KCI	Compare count immediate	25
1011	LVS	Load value with sum	28

UNCONDITIONAL BRANCHING; Instruction bits 19-31

xxx000000iiii	Mnemonic	Name	Page
Operations			
010	B	Branch	39
011	BR	Branch relative	40
000	BE	Branch enabled	40
001	BD	Branch disabled	40
100	BEW	Branch enabled and wait	40
110	NOP	No operation	40

INDICATOR BRANCHING; Instruction bits 25-31

1000xxi	Mnemonic	Name	Page
Operation	BI	Branch on indicator	40
Modifiers			
0		Leave indicator	
1		Set indicator to zero	
0		Branch if off	
1		Branch if on	

BIT BRANCHING; Instruction bits 51-63

111000000xxxi	Mnemonic	Name	Page
Operation	BB	Branch on bit	41
Modifiers			
0		Leave bit	
1		Invert bit	
0		Leave bit	
1		Set bit to zero	
0		Branch if off	
1		Branch if on	

INDEX BRANCHING; Instruction bits 19-31

jjjjxx1001xxi	Mnemonic	Name	Page
Operations			
0	CB	Count and branch	26,40
1	CBR	Count, branch and refill	26,41
Modifiers			
0		Branch if count non-zero	
1		Branch if count zero	
00		Leave value unchanged	
01		Add half to value	
10		Add one to value	
11		Subtract one from value	

STORE INSTRUCTION COUNTER IF; Instruction bits 24-31

1000iiii	Mnemonic	Name	Page
Operation	SIC	Store instruction counter if	41

INPUT-OUTPUT OPERATIONS; Instruction bits 51-63

xxxx10000iiii	Mnemonic	Name	Page
Operations			
0000	RD	Read	119
0001	W	Write	119
0010	CTL	Control	120
0011	LOC	Locate	120
1001	REL	Release	121
0100	RD(SEOP)	Read SEOP	121
0101	W(SEOP)	Write SEOP	121
0110	CTL(SEOP)	Control SEOP	121
0111	LOC(SEOP)	Locate SEOP	121
1101	REL(SEOP)	Release SEOP	121
1000	CCW	Copy control word	122

TRANSMIT OPERATIONS; Instruction bits 51-63

jjjjfdx10iiii	Mnemonic	Name	Page
Operations			
0	T	Transmit	30
1	SWAP	Swap	30
Modifiers			
0		Forward	
1		Backward	
0		Direct count	
1		Immediate count	

MISCELLANEOUS OPERATIONS; Instruction bits 19-31

xxx100000iiii	Mnemonic	Name	Page
Operations			
000	R	Refill	26
001	RCZ	Refill on count zero	26
010	EX	Execute	50
011	EXIC	Execute indirect and count	50
100	Z	Store zero	31

INDEXING MODES FOR VARIABLE FIELD LENGTH OPERATIONS; Instruction bits 32-34

ppp	Mnemonic	Name	Page
		Normal address modification	24
000		Direct addressing	
100	I(suffix)	Immediate addressing	
		Progressive indexing	21
001	V+	Add to value	
010	V+C	Add to value and count	
011	V+CR	Add to value, count and refill	
101	V-	Subtract from value	
110	V-C	Subtract from value and count	
111	V-CR	Subtract from value, count, and refill	

Powers of Two

2^n	n	2^n	n
2	1	8 589 934 592	33
4	2	17 179 869 184	34
8	3	34 359 738 368	35
16	4	68 719 476 736	36
32	5	137 438 953 472	37
64	6	274 877 906 944	38
128	7	549 755 813 888	39
256	8	1 099 511 627 776	40
512	9	2 199 023 255 552	41
1 024	10	4 398 046 511 104	42
2 048	11	8 796 093 022 208	43
4 096	12	17 592 186 044 416	44
8 192	13	35 184 372 088 832	45
16 384	14	70 368 744 177 664	46
32 768	15	140 737 488 355 328	47
65 536	16	281 474 976 710 656	48
131 072	17	562 949 953 421 312	49
262 144	18	1 125 899 906 842 624	50
524 288	19	2 251 799 813 685 248	51
1 048 576	20	4 503 599 627 370 496	52
2 097 152	21	9 007 199 254 740 992	53
4 194 304	22	18 014 398 509 481 984	54
8 388 608	23	36 028 797 018 963 968	55
16 777 216	24	72 057 594 037 927 936	56
33 554 432	25	144 115 188 075 855 872	57
67 108 864	26	288 230 376 151 711 744	58
134 217 728	27	576 460 752 303 423 488	59
268 435 456	28	1 152 921 504 606 846 976	60
536 870 912	29	2 305 843 009 213 693 952	61
1 073 741 824	30	4 611 686 018 427 387 904	62
2 147 483 648	31	9 223 372 036 854 775 808	63
4 294 967 296	32	18 446 744 073 709 551 616	64

Index